

Künstliche Intelligenz in der Kompositionslehre

Eine Untersuchung von Long-Short-Term-Memory-Netzen
zur Analyse von Kontrapunkten nach Fux

Bachelorarbeit Informatik

vorgelegt von

Michael Jordan Scholzen

Matr. 11112624

Erstgutachter: Prof. Dr. rer. nat. Wolfgang Konen (TH Köln)

Zweitgutachter: Prof. Dr. Lutz Köhler (TH Köln)

5. Dezember 2019

Künstliche Intelligenz in der Kompositionslehre

Eine Untersuchung von Long-Short-Term-Memory-Netzen
zur Analyse von Kontrapunkten nach Fux

Inhaltsverzeichnis

Einleitung	5
1 Künstliche Intelligenz in der Musiktheorie	6
2 Die Kontrapunktregeln	8
2.1 Die Analyse	8
2.2 Bewertung nach den Regeln	10
3 CounterPai	11
3.1 Erweiterung des Evaluators	12
4 Trainingsdaten	15
4.1 Handgeschriebene Daten	15
4.2 Mutations-Kontrapunktgenerator	16
4.3 Stochastischer-Kontrapunktgenerator	16
4.3.1 c.f.	16
4.3.2 c.p.	17
4.4 Analyse der generierten Daten	18
4.4.1 Postseparierte Mutationsdaten	18
4.4.2 Präseparierte Mutationsdaten	18
4.4.3 Stochastische Daten	19
4.4.4 Vergleich der Daten	19
4.5 Repräsentation der Noten	22
4.5.1 Gleiche Datensätze	24
5 Verwendete Netze	28
5.1 Das Feedforward-Netz	28
5.2 Das Long-Short-Term-Memory-Netz	29
5.3 Dropout	30
5.4 Loss-Functions	30
5.5 Optimizer	30
6 Bewertung der Kontrapunktregeln	31
6.1 Gewählte Inputs	31
6.1.1 Feedforward-Input	31
6.1.2 Durchlaufender sequenzieller Input	31
6.1.3 Zusammenlaufender sequenzieller Input	32
6.1.4 Bewertung der Netze	32

6.2	Training mit prä- und postseparierten Mutationsdaten.....	34
6.3	Training mit stochastischen Trainingsdaten.....	36
6.4	Bewertung mehrerer Regeln.....	38
6.5	Reduzieren der Trainingsbeispiele	40
6.6	Diskussion der Ergebnisse.....	44
	Fazit.....	47
	Ausblick	48
	Literaturverzeichnis.....	49
	Abbildungsverzeichnis	51
	Glossar.....	57
	Nomenklatur.....	58
	Anhang	59
A	Ergebnisse	59
A1	Vollständige Analyse der generierten Daten.....	59
A1.1	Trainingsdaten	59
A1.2	Testdaten	61
A1.3	Validationsdaten.....	63
A1.4	Analyse auf gleiche Datensätze.....	65
A2	Versuchsreihe „Training mit postseparierten Mutationsdaten“.....	66
A3	Versuchsreihe „Training mit präseparierten Mutationsdaten“	69
A4	Versuchsreihe „Training mit stochastischen Daten“	71
A5	Versuchsreihe „Training einzelner Netze auf alle Regeln“	74
A6	Versuchsreihe „Reduzieren der Trainingsbeispiele“	78
B	Ausschnitt 1 aus der Dokumentation der vorangegangenen Arbeit	81
B1	Bewertung der Noten.....	81
B1.1	Annahmen über die Wahrscheinlichkeiten der Noten.....	83
B1.2	R1 Start- und Endregel	83
B1.3	R2 Bevorzugte Motion	83
B1.4	R3 Bevorzugte Intervalle.....	84
B1.5	R4 Einleiten der Schlussnote.....	85
B1.6	R5 Bevorzugte Motion-abhängige Intervalls.....	86
B1.7	R6 Verbotene Skips.....	87
B1.8	R7 Ties nur auf dem Down Beat	87

B1.9	R8 Verbotene Suspensions	88
B1.10	R9 Achtelnoten nur auf Weak Beats	88
B1.11	Beispiel anhand eines 1. Species Counterpoint	89
C	Ausschnitt 2 aus der Dokumentation der vorangegangenen Arbeit	91
C1	Aufbau von CounterPai	91
C1.1	Sheet_To_Audio.....	92
C1.2	GUI.....	96
C1.3	Utility.....	97
C1.4	Evaluator	98
	Eidesstattliche Erklärung.....	99

Einleitung

Musik zu komponieren ist kein leichtes Unterfangen und sollte gelernt sein, wenn dabei etwas Wohlklingendes entstehen soll. Darüber, was genau „wohlklingend“ in diesem Zusammenhang bedeutet, sind sich nicht alle einig, dennoch gibt es grundlegende Regeln, die eine breite Zustimmung erhalten.

Die westliche Kompositionslehre versucht, Konzepte zu vermitteln, die es dem Musikstudenten¹ ermöglichen, etwas zu komponieren, das von den meisten Menschen als wohlklingend wahrgenommen wird. Einen großen Einfluss auf die Lehre hatte ein im Jahr 1725 veröffentlichtes Werk von Johann Joseph Fux, der *Gradus ad Parnassum*. Dieses Buch, das grundlegende Regeln vermittelt, wie Musik geschrieben werden kann, hat bis heute einen großen Einfluss auf die Kompositionslehre.

Der *Gradus ad Parnassum* liefert ein Grundgerüst, Melodien in mehreren Stimmen gegeneinander zu setzen, er beantwortet Fragen nach der Harmonie von Noten, die gleichzeitig erklingen, sowie nach der Stimmführung, die dafür sorgt, dass zwei zur selben Zeit gespielte Melodien voneinander unterschieden werden können.

Fux beschreibt außerdem ein Lehrprinzip, welches dieses Grundgerüst vermitteln soll. Der Lehrende gibt eine Melodie vor, den *cantus firmus*, und der Musikstudent versucht, eine gut klingende Melodie darüber zu legen, den Kontrapunkt. Der Lehrende überprüft anschließend das vom Musikstudenten erstellte Stück auf die im *Gradus ad Parnassum* beschriebenen Regeln des Kontrapunkts – der Musikstudent soll so aus seinen Fehlern lernen.

Die im *Gradus ad Parnassum* beschriebenen Regeln sind nicht trivial, aber für einen Musikstudenten im Kompositionsstudium meist in nur einem Semester erlernbar. Ob diese Regeln auch für eine künstliche Intelligenz (KI) lernbar sind, soll in dieser Arbeit untersucht werden. Es soll geprüft werden, ob eine KI, die die Regeln gelernt hat, verlässlich genug ist, um anschließend die Aufgabe des Lehrenden zu übernehmen. Solch eine KI könnte die von einem Musikstudenten geschriebenen Kontrapunkte auf Fehler überprüfen und damit den Lernprozess unterstützen.

In Musikstücken folgt ein Takt dem anderen. Diese Arbeit soll zeigen, ob dieser sequenzielle Aufbau einem Long-Short-Term-Memory einen Vorteil beim Lernen verschafft.

¹ Ein Student im Kompositions- oder einem sonstigen Studium, das die Komposition unterrichtet.

1 Künstliche Intelligenz in der Musiktheorie

In der KI-Forschung stellt die Musiktheorie ähnliche Anforderungen wie die Computerlinguistik: die Musik kann einerseits durch Token beschrieben werden und andererseits können Regeln definiert werden, wie mit diesen Token umzugehen ist. Ob diese Regeln allgemeingültig sind, kann aber nicht immer exakt bestimmt werden. [Meehan, 1980]

Die Forschung lässt sich dabei grob in die Bereiche generierende und bewertende KIs unterteilen.

Auf Seite der generierenden KI-Systeme wurden in [Farboor & Schoner, 2001] auf Markov-Chains basierende Versuche unternommen, eine zweite Melodie über eine vorgegebene zu legen. Die Autoren gaben in ihrer Arbeit musikalische Konzepte formuliert als abhängige Wahrscheinlichkeiten vor und ließen die zugehörigen Wahrscheinlichkeitstabellen dann anhand von Beispieldaten füllen. Die daraus resultierende KI konnte laut den Autoren nicht nur ihren Regeln konforme Stimmen generieren, sondern war in den Ergebnissen sogar vergleichbar mit denen eines fachkundigen Musikers.²

In [Liang, 2016] konnte gezeigt werden, dass ein Long-Short-Term-Memory-Netz mit Bach-Chorälen soweit trainiert werden kann, dass es selbst Bach-ähnliche Musikstücke erzeugen kann, wenn ihm eine Einstiegssequenz vorgegeben wird. Eine im Rahmen der Arbeit durchgeführte Studie kam zu dem Schluss, dass in nur 59% der Fälle die von der KI geschriebenen Stücke von Originalen unterschieden werden konnten. In der Arbeit wurden auch Versuche unternommen, die aktiven Neuronen des Netzes auf ihre Funktion zur Erkennung von musikalischen Strukturen zu untersuchen. Es konnten so Vermutungen angestellt werden, dass das Netz unter anderem in der Lage ist, den Grundakkord in einer Inversion zu erkennen. In [Dada, 2018] konnten ähnliche Ergebnisse erzielt werden.

Auch auf kommerzieller Seite sind die generierenden KI-Systeme inzwischen etabliert. Plugins wie Musico³ erlauben Musikern, Melodien zur Verwendung oder Inspiration zu generieren. Programme wie AIVA⁴ erlauben sogar das Generieren ganzer Musikstücke.

Die bewertenden KI-Systeme werden in der Literatur teils auch disziplinübergreifend als AACs (*artificial art critics*) bezeichnet.

In [Manaris, et al., 2007] konnte gezeigt werden, dass ein genetischer Algorithmus, dessen Fitnessfunktion durch einen AAC gesteuert wurde, ansprechende Musik erzeugen kann. In der Arbeit wurden aus einem Onlinearchiv Musikstücke anhand der monatlichen Zugriffe ausgewählt. Die so ausgewählten Stücke wurden in populäre (viele Zugriffe) und unpopuläre (wenige Zugriffe) aufgeteilt

² „Not only does the composed line comply with the strict rules of sixteenth-century counterpoint, but the results are also musical and comparable to those created by a knowledgeable musician“ [Farboor & Schoner, 2001, S. 4]

³ <https://www.musi-co.com> (13.11.2019) Musico generiert Melodien und Rhythmen. Es richtet sich vor allem an Komponisten aus dem Bereich der elektronischen Musik.

⁴ <https://www.aiva.ai/> (13.11.2019) AIVA generiert ganze Musikstücke. Die Zielgruppe sind Unternehmen, die Ambientmusik oder Werbemusik generieren wollen. Leider lassen sich zu der verwendeten Technik keine wissenschaftlichen Artikel finden. Die unter anderem in einem TED-Talk vorgestellten Ergebnisse des Programms sind allerdings beeindruckend.

https://www.ted.com/talks/pierre_barreau_how_ai_could_compose_a_personalized_soundtrack_to_your_life (21.11.2019)

und mit ihnen neuronale Netze trainiert. Die Musikstücke wurden den Netzen dabei nicht als Notenfolgen, sondern als vorher extrahierte Informationen über die Stücke übergeben.⁵

In [Holland, 2000] wird das Lernprogramm Lasso beschrieben, das ohne KI-Komponente arbeitet. Es erlaubt, ähnlich dem in dieser Arbeit verwendeten Programm, die Eingabe eines zweistimmigen Kontrapunkts und kennzeichnet anschließend die Noten entsprechend vorher festgelegter Regeln. Allerdings wurden dort 127 sehr spezielle Regeln formuliert, und der ursprüngliche Autor [Newcomb, 1985] gab an, dass die Musikstudenten sich durch die Quantität überfordert fühlten.

Die in dieser Arbeit untersuchten neuronalen Netze sollen auf einzelne Regeln trainiert werden. So soll einerseits ermöglicht werden, dass weitere Regeln hinzugefügt werden können und andererseits die Möglichkeit geschaffen werden, für jede Regel ein Feedback in Form einer Nachricht zu hinterlassen. Hiermit kann dem Musikstudenten für Regeln, die unter Umständen gebrochen werden dürfen, eine entsprechende Nachricht angezeigt werden.

⁵ Es wird unter anderem eine Metrik beschrieben, die die Anzahl der Intervalle im Stück angibt „[...] 168 half steps, 86 unisons, 53 whole steps [...]“ [Manaris, et al., 2007]

2 Die Kontrapunktregeln

Dieses Kapitel soll beschreiben, was die Kontrapunktregeln sind, und ein Gefühl dafür vermitteln, was neuronale Netze leisten müssen, um sie erkennen zu können.

Im *Gradus ad Parnassum* werden die Regeln in einem Dialog vermittelt. Es gibt keine klare Abgrenzung zwischen den Regeln, sie werden nach und nach im Gespräch zwischen Schüler und Lehrer eingeführt. Teils stehen sie zu Beginn eines Kapitels und werden durch den Lehrmeister erklärt, teils werden sie erst ersichtlich, wenn der Schüler sein Werk erklärt oder vom Meister auf einen Fehler aufmerksam gemacht wird. (Abb. 1)

In einer vorangegangenen Arbeit⁶ wurden die Regeln aus dem Text abgeleitet und mit Namen versehen, um sie bewerten zu können.

Die Regeln beziehen sich auf eine grundlegende musiktheoretische Analyse der Noten.

finally, I proceeded according to the rule which says: from an imperfect consonance to a perfect consonance one must move in contrary motion. The eleventh pair of notes, the conclusion, is, as you

Abb. 1 Dialog im *Gradus ad Parnassum*. Der Schüler Joseph erklärt dem Meister Aloyse, was er sich beim Erstellen seines Stücks gedacht hat.

2.1 Die Analyse

Die Analyse bezieht sich auf die Beziehung der durch den Musikstudenten eingegebenen Noten, dem Kontrapunkt (*c.p.*⁷), untereinander sowie auf deren Beziehung zu den vom Lehrer vorgeschriebenen Noten, dem Cantus Firmus (*c.f.*).

Für alle Noten des *c.p.* wird ermittelt, mit welcher Note des *c.f.* sie zusammen erklingen. Dieses gleichzeitige Erklingen der Noten erzeugt ein Intervall (Abb. 2). Intervalle können dissonant (*DisI*), imperfekt konsonant (*iPerCI*⁸) oder perfekt konsonant (*PerCI*) sein.

Diese Eigenschaften sind per Definition festgelegt:

DisI	2, 4, 7
iPerCI	3, 6
PerCI	1, 5, 8

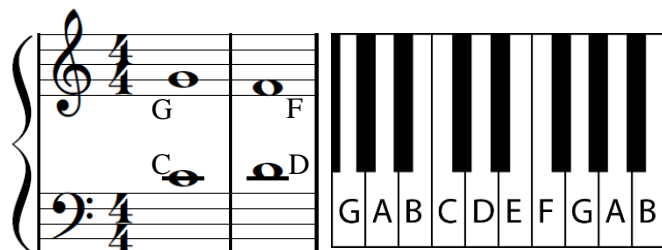


Abb. 2 Intervalle werden immer einschließlich der untersuchten Noten berechnet. Für die Intervalle zwischen *c.f.* und *c.p.*: von C nach G ist das Intervall 5, von D nach F 3. Für die Sprünge in der oberen Stimme: G nach F 1. In der unteren Stimme: C nach D ebenfalls 1.

Beziehen sich die Regeln nach Fux auf die Intervalle zwischen den Noten der jeweiligen Stimmen, werden die Intervalle als Notensprünge (*S*⁹) bezeichnet, und es wird nicht zwischen dissonanten und konsonanten Sprungintervallen unterschieden.

⁶ Die vorangegangene Arbeit ist im Rahmen des Praxisprojekts erstellt worden und dokumentiert die Vorgehensweise beim Formulieren der Kontrapunktregeln sowie den Aufbau eines Computerprogramms namens „CounterPai“, das die Eingabe und Bewertung von Kontrapunkten ermöglicht. Einige Teile der Dokumentation sind im Anhang dieser Arbeit zu finden; auf sie wird an den relevanten Stellen mit Infokästen hingewiesen. Die vollständige Dokumentation ist unter diesem Link abrufbar:

https://github.com/Ni2Be/CounterPAI/releases/download/BETA_v0.7.0/Dokumentation.CounterPai.pdf (04.12.2019)

⁷ Engl.: *counterpoint*.

⁸ Engl.: *imperfect consonant interval*.

⁹ Engl.: *skip*.

Weitere Regeln nach Fux beziehen sich auf die Bewegung der Noten (Abb. 3). Bewegen sich vier Noten voneinander in entgegengesetzte Richtung, wird dies als Gegenbewegung (**CoMo**¹⁰) bezeichnet. Bewegen sie sich in die gleiche Richtung, ist von Gleichbewegung (**SiMo**¹¹) die Rede. Bleibt eine Note auf derselben Höhe und nur die andere ändert ihre Position, ist das eine Seitenbewegung (**ObMo**¹²).

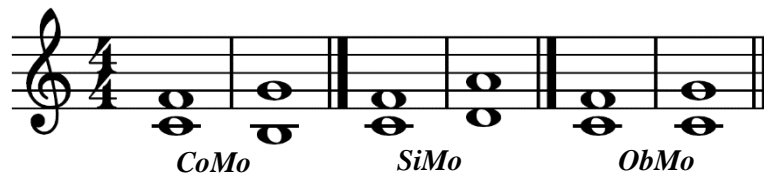


Abb. 3 Verschiedene Arten der Bewegung.

Es gibt außerdem Regeln, die sich darauf beziehen, in welchem Takt eine Note steht (Abb. 4). Zum Beispiel ist eine Pause nur im ersten Takt (**FBar**) erlaubt. Die Regeln im Allgemeinen beziehen sich meist auf die mittleren Takte (**MBar**). Für das Einleiten des Endes (**BLBBar**) und das Ende (**LBar**) gibt es spezielle Regeln, die nur in diesen Takten gelten.

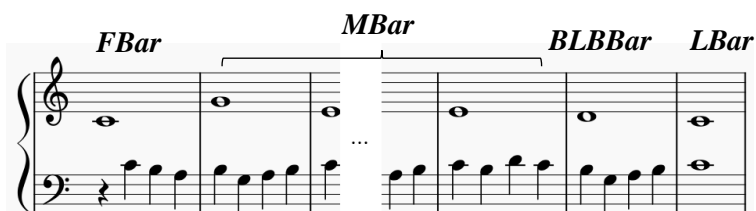


Abb. 4 Taktpositionen.

In den einzelnen Takten findet wieder eine Unterscheidung statt, wo die Note platziert ist. Dabei gelten für die erste Position im Takt, den **Downbeat (DBeat)**, besonders strenge Regeln, da hier ein dissonantes Intervall schnell schlecht klingen kann. Für den **Upbeat (UBeat)** gelten weniger Regeln, und für die beiden **Weakbeats (WBeat)** gelten nur einzelne spezielle Regeln.

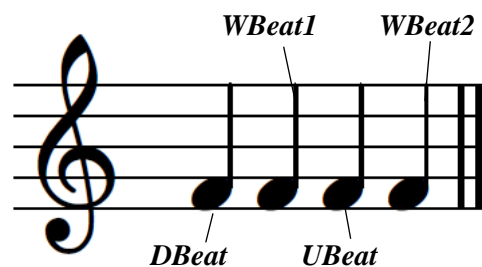


Abb. 5 Beatpositionen.

Die Regeln beziehen sich zusätzlich zu der Analyse der Noten auch auf die Eigenschaften einer Note. Eine Note kann gehalten (**Tied**) sein, und der Wert (**Value**) einer Note kann eine Rolle spielen. (Abb. 6, Abb. 7)



Abb. 6 Ein gehaltenes C im zweiten Takt. Die Anfrage **Tied(N)** würde das Ergebnis „True“ liefern.



Abb. 7 Die Anfrage **Value(N)** würde von links nach rechts die Ergebnisse „halbe Note“, „Viertelnote“, „Achtelnote“, „Achtelnote“ liefern.

¹⁰ Engl.: *Contrary Motion*.

¹¹ Engl.: *Similar Motion*.

¹² Engl.: *Oblique Motion*.

2.2 Bewertung nach den Regeln

In der vorangegangenen Arbeit wurden die Regeln als Wahrscheinlichkeitstabellen formuliert. Dabei wurden Wahrscheinlichkeitswerte für das Verletzen einer Regel angenommen, die den Wortlaut des Textes repräsentieren sollen. Textstellen wie „[...] more imperfect than perfect consonances should be employed“ führen dazu, dass der Wert von Noten, die die Eigenschaft *iPerCI* erfüllen, eine höhere Wahrscheinlichkeit haben als die, die die Eigenschaft *PerCI* erfüllen. Sätze wie „[...] from an imperfect consonance to a perfect consonance one must go in contrary motion“ führen dazu, dass eine Wahrscheinlichkeit den Wert

0,0 annimmt, wenn sie für eine Note unter den gegebenen Randbedingungen nicht erfüllt ist.¹³

Die Regeln reichen von simpel bis recht komplex. Für Regel 1 „Start- und Endregel“ muss nur die Vorbedingung erfüllt sein,

dass die Taktposition der erste oder letzte Takt ist, und danach auf das Intervall zur Note im *c.f.* geprüft werden. Andere Regeln wie Regel 3 „Bevorzugte Intervalle“ müssen aber Informationen über die umliegenden Noten sammeln und diese Informationen mit logischen Operatoren verknüpfen.

Die neuronalen Netze, die im Laufe dieser Arbeit untersucht werden, sollen ohne Vorwissen die Analyse der Noten sowie das Prüfen auf die Regeln erlernen.

Bevorzugte <i>Motions</i>	Erzeugte <i>Motion</i>	Wahrscheinlichkeit
$P(M Pos(N) = DBeat)$	<i>CoMo</i>	1,0
	<i>DiMo</i>	0,2
	<i>ObMo</i>	0,9

Abb. 8 Die Tabelle von Regel 2 „Bevorzugte Motion“. Mit welcher Bewegung eine Note erreicht wird, spielt nur auf dem Downbeat eine Rolle, dargestellt durch die Notation $Pos(N) = DBeat$.

R1 Start- und Endregel	$P(I_t N_{t-1} = \emptyset \vee N_{t+1} = \emptyset)$
R2 Bevorzugte <i>Motion</i>	$P(M Pos(N) = DBeat)$
R3 Bevorzugte <i>Intervals</i>	$P(I_t Tied(N_t) \wedge Pos(N_t) \wedge I_{t+1} \wedge S_{t+1} \wedge N_{t-1} \wedge N_{t+1})$
R4 Einleiten der Schlussnote	$P(I_t N_{t+2} = \emptyset \wedge bass(c.f.)),$ $P(I_t N_{t+3} = \emptyset \wedge bass(c.f.) = true)$
R5 Bevorzugte <i>Motion</i> -abhängige <i>Intervals</i>	$P(I_t M_t \wedge I_{prevDBeat})$
R6 Verbotene <i>Skips</i>	$P(S I_{t-1})$
R7 Ties nur auf dem <i>Downbeat</i>	$P(Tied(N) Pos(N))$
R8 Verbotene <i>Suspensions</i>	$P(I_t Tied(N_t) \wedge Voice(c.f.) \wedge I_{t-1})$
R9 Achtelnoten nur auf <i>Weakbeats</i>	$P(N Value(N) \wedge Pos(N) \wedge (S_t \wedge S_{t+1}))$

Abb. 9 Die Kontrapunktregeln. R5 wurde um eine Teilregel erweitert und hängt jetzt auch von der vorherigen DBeat-Note ab.

Das Glossar auf Seite 57 erläutert die in diesem Kapitel verwendeten Begriffe und Abkürzungen.

Im Anhang ist ein Ausschnitt zu finden, der die Regeln mit den zugehörigen Zitaten aus Fux' Buch sowie den Tabellen auflistet. Es wird außerdem anhand eines Beispiels erklärt, wie eine Note bewertet wird. (Ausschnitt 1 aus der Dokumentation der vorangegangenen Arbeit S. 81)

¹³ Für das Trainieren der neuronalen Netze spielt diese Wahrscheinlichkeit allerdings keine Rolle. Es wird eine harte Codierung verwendet, die nur aussagt, ob eine Regel verletzt wurde oder nicht.

3 CounterPai

Das Programm *CounterPai*¹⁴ (Abb. 10) wurde im Rahmen der vorherigen Arbeit entwickelt. Es wird in dieser Arbeit dazu genutzt, Kontrapunkte zu generieren und zu bewerten. Aus Usersicht erlaubt das Programm das Eingeben von Noten und gibt anschließend ein visuelles Feedback, ob Kontrapunktregeln verletzt wurden, und gibt Informationen zu den Noten an, wie sie im vorherigen Kapitel beschrieben wurden.

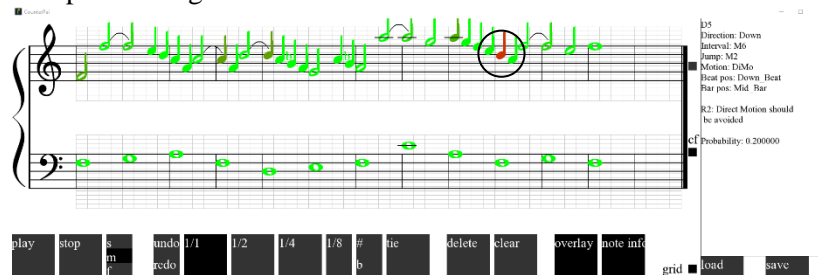


Abb. 10 CounterPai v0.7.0. Die Informationen rechts beziehen sich auf die umkreiste Note.

CounterPai teilt sich aus Entwicklerperspektive in vier Komponenten (Abb. 11) auf:

Sheet_To_Audio

Sheet_To_Audio ist eine Komponente, die es ermöglicht, Notenblätter (*Sheets*) zu verändern, Noten hinzuzufügen oder zu löschen, und diese Notenblätter auf den Lautsprechern eines Computers wiederzugeben.

GUI

Diese Komponente erlaubt es dem User, Notenblätter zu manipulieren, zu laden und zu speichern.

Utility

Die Utility bildet eine Abstraktionsschicht für plattformabhängige Funktionen wie den Windows-Ordnerdialog.

Evaluator

Die Komponente Evaluator steht in dieser Arbeit im Vordergrund. Sie analysiert die Noten eines Notenblatts und bewertet die Noten anhand der Kontrapunktregeln.

Im Rahmen dieser Arbeit wird nur die Komponente Evaluator erweitert.

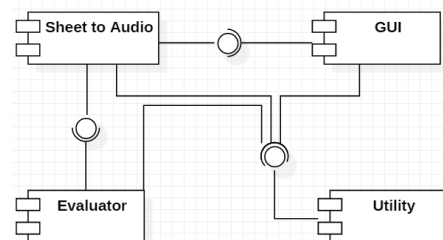


Abb. 11 CounterPai-Komponentendiagramm.

Im Anhang ist ein Ausschnitt zu finden, der genauer auf die Komponenten eingeht, die im Rahmen der vorangegangenen Arbeit implementiert wurden. (Sheet_To_Audio, GUI, Utility, Evaluator (vor Erweiterung))

(Ausschnitt 2 aus der Dokumentation der vorangegangenen Arbeit S. 91)

¹⁴ Aus dem englischen Wort für Kontrapunkt „*counterpoint*“ und *artificial intelligence* „*ai*“.

Das Programm und der Sourcecode (inklusive der in dieser Arbeit implementierten KI-Komponente) sind unter <https://github.com/Ni2Be/CounterPAI> abrufbar. (04.12.2019)

3.1 Erweiterung des Evaluators

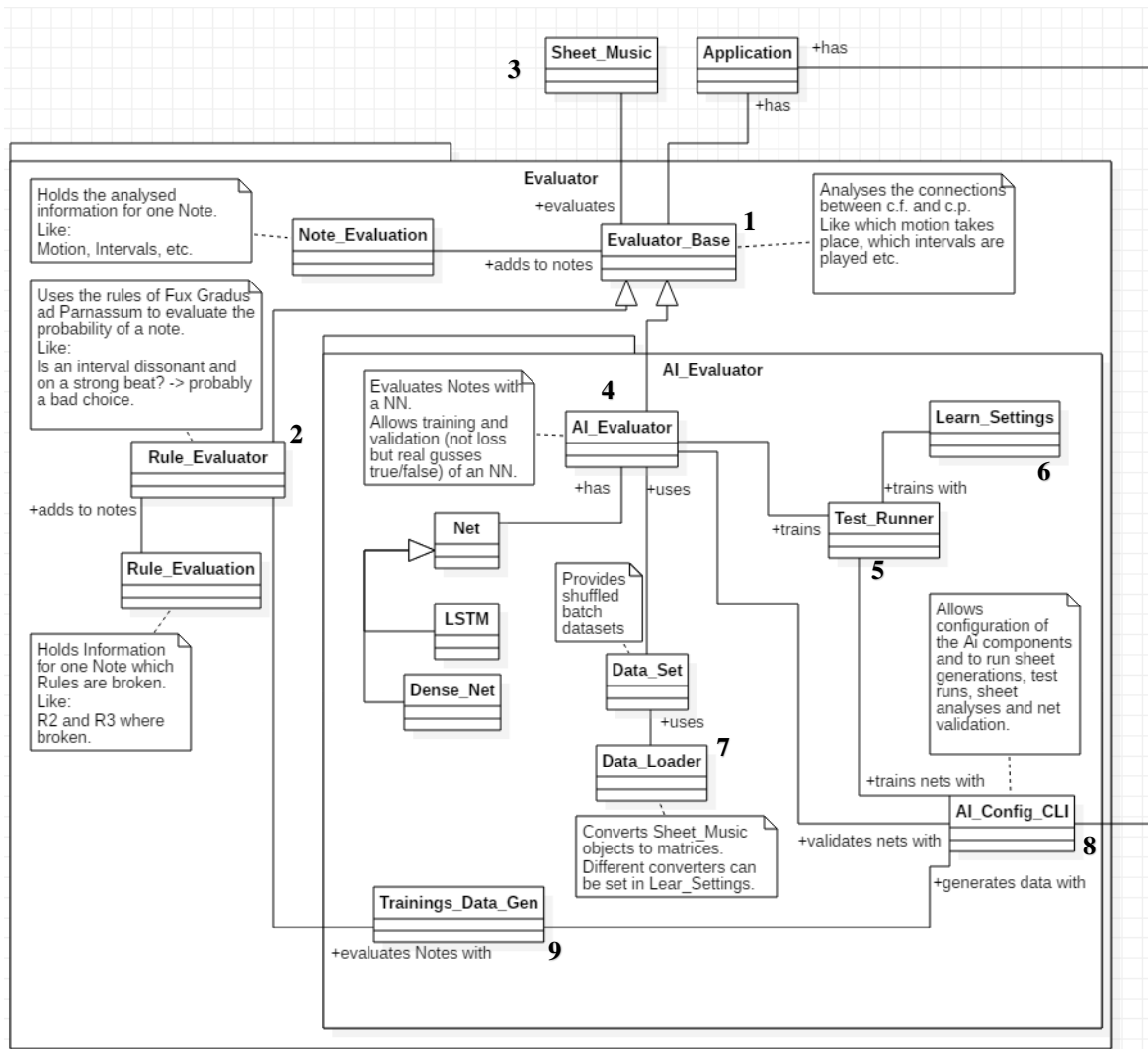


Abb. 12 Komponente Evaluator mit internen Klassen.

Die Komponente Evaluator wurde um die interne Komponente AI_Evaluator erweitert. Die Klassen Evaluator_Base (Abb. 12, 1) und Rule_Evaluator (2) sind im Rahmen der vorangegangenen Arbeit erstellt worden. Sie beinhalten die Logik, Noten zu analysieren und nach den Regeln bewerten zu können. Sie sind so aufgebaut, dass ihnen ein Sheet_Music-Objekt (3) (also das Notenblatt) übergeben werden kann und sie dann jeder Note eine Nachricht mit ihrer Bewertung anhängen. (Abb. 13)

```
{"NOTE_EVALUATION": "p: Mid_Bar b: Down_Beat j: P8 d: Up m: DiMo i: M6",
"RULE_EVALUATION": "prb: 0.5 R2"}
```

Abb. 13 Ausschnitt einer sheet-Datei mit angehängten Nachrichten.

Die Klasse AI_Evaluator (Abb. 12, 4) soll an die Stelle treten, an der im

```
"AI_EVALUATION": "R2 R4"
```

Abb. 14 Nachricht des AI_Evaluators der annimmt, dass Regel 2 und 4 gebrochen wurden.

Moment der Rule_Evaluator steht, nur sollen die Noten statt durch Analysieren und Prüfen durch Algorithmen, alleine durch ein neuronales Netz geprüft werden. (Abb. 14)

Der AI_Evaluator wird mithilfe der Klasse Test_Runner (Abb. 12, 5) trainiert. Der Test_Runner lädt die Learn_Settings (6) und startet anschließend das Training des AI_Evaluators. Als Dateiformat der Learn_Settings wurde eine JSON-Formatierung gewählt. (Abb. 15)

Sind die Settings vom Test_Runner eingelesen worden, wird der AI_Evaluator mit den entsprechenden Einstellungen trainiert. Die Information der Settings „nn_type“ bestimmt, welche Art Netz trainiert werden soll und die Zeile „data_converter“ bestimmt, wie die Sheet_Music-Objekte vom Data_Loader (Abb. 12, 7) zu Inputfeatures konvertiert werden. „data_converter_info“ bestimmt dabei, auf welche Regel ein Netz trainiert werden soll. Entsprechende *Target values* werden vom Data_Loader produziert.

Das AI_Config_CLI (8) ist eine Konsolenoberfläche, die das Steuern des Trainingsprozesses ermöglicht. (Abb. 16)

Das CLI¹⁵ ist als Baum konzipiert und bietet die folgenden Funktionen:

- run tests → startet alle im test_settings-Ordner vorhandenen Tests (Test bedeutet dabei Training eines NNs¹⁶ mit einer Learn_Settings-Datei)
- generate sheets
 - o alter sheets count → erlaubt das Einstellen der Anzahl von generierten Sheets (z.B. 10.000 Trainings-Sheets, 2.000 Test-Sheets, 2.000 Validation-Sheets)
 - o switch generators → erlaubt das Einstellen der verschiedenen Generatoren. (z.B. den stochastischen Generator)
 - o change folder names → erlaubt das Ändern der Ordner, in denen die generierten Daten gespeichert werden sollen
 - o generate sheets → startet das Generieren
 - o exit generate trainings sheets dialog → kehrt zum AI-Config-Dialog-Hauptbildschirm zurück.
- analyse sheets
 - o analyse → startet die Analyse der Trainings-, Test- und Validation-Sheets (Gibt eine Statistik darüber aus, wie oft Noten vorkommen, wie oft welche Regel verletzt wurde etc.)
 - o change folder names → erlaubt das Ändern der Ordner, in denen die zu analysierenden Sheet-Dateien liegen.
 - o exit analyse sheets dialog → kehrt zum AI-Config-Dialog-Hauptbildschirm zurück.

```
{
  "name":
  "R1_evaluate_fux_rules_from_two_sides_dense_NN_1",
  "nn_type": "DENSE_NN",
  "in_size": 680,
  "hidden_size": 312,
  "out_size": 1,
  "hidden_layer_count": 3,
  "batch_size": 50,
  "epochs": 30,
  "dropout": 0.0,
  "optimizer": "ADAM",
  "loss_func": "BCEL",
  "learning_rate": 0.0001,
  "train_data_folder":
  "data/trainings_data/stoch/train",
  "test_data_folder":
  "data/trainings_data/postsep/test",
  "valid_data_folder":
  "data/trainings_data/postsep/valid",
  "data_converter":
  "evaluate_fux_rules_from_two_sides_dense_NN_1",
  "data_converter_info": "R1",
  :
}
```

Abb. 15 Ausschnitt einer Learn_Settings-Datei. Hinter der „data_converter_info“ sind außerdem Angaben über die möglichen „nn_typen“, „optimizer“, etc. gespeichert.

```
-----AI Config Dialog-----
options:
"r": [r]un tests
"g": [g]enerate sheets
"a": [a]nalyse sheets
"v": [v]alidate net
"e": [e]xit ai config dialog
```

Abb. 16 Hauptbildschirm des AI_Config_CLI.

¹⁵ command-line interface

¹⁶ Neuronales Netz

- validate net
 - validate one net → erlaubt das Navigieren durch Systemordner und das Auswählen eines Models (trainiertes Netz), das geprüft werden soll. Gibt anschließend eine *Confusion Matrix* aus.
 - batch validation → erlaubt das Navigieren durch Systemordner und validiert anschließend alle Models im ausgewählten Ordner und allen Unterordnern. Speichert die *Confusion Matrices* in CSV-Dateien in den jeweiligen Ordnern.
 - exit validation dialog → kehrt zum AI-Config-Dialog-Hauptbildschirm zurück.
- exit ai config dialog → verlässt den AI-Config-Dialog-Hauptbildschirm und startet CounterPai.

Wird im CLI die Option „generate sheets“ gewählt, werden mithilfe der Klasse `Trainings_Data_Gen` (Abb. 12, 9) entsprechend der Einstellungen Sheets generiert, mithilfe des `Rule_Evaluators` bewertet und anschließend abgespeichert. Diese generierten Sheet-Dateien werden verwendet, um die neuronalen Netze zu trainieren. Im folgenden Kapitel werden die verschiedenen Generatoren diskutiert.

4 Trainingsdaten

Da keine Datenbanken mit bewerteten Kontrapunktnotenblättern gefunden werden konnten, wurden verschiedene Methoden untersucht, Trainingsdaten zu generieren.

Alle generierten Daten werden durch den Rule_Evaluator des Programms CounterPai bewertet. Der Rule_Evaluator wendet die Kontrapunktregeln nach Fux an, um den *c.p.*-Teil eines Kontrapunkts zu bewerten (der *c.f.*-Teil ist vom Lehrer vorgegeben und muss nicht bewertet werden). Jede *c.p.*-Note wird auf die Kontrapunktregeln geprüft, und die Note wird entsprechend markiert. (Abb. 14) Ein so bewertetes Notenblatt wird anschließend abgespeichert.

Für die Trainingsdaten wurde eine Einschränkung gewählt, die die Menge der möglichen Kontrapunkte einschränkt. Für die obere Stimme werden nur Noten vom unteren A3 bis zum oberen C6, für die untere nur Noten vom unteren C2 bis zum oberen E4 für die Eingabe und Generierung der Daten zugelassen. Das Intervall ist in beiden Fällen 17. Da einige Noten auch um einen Halbton erhöht, bzw. vermindert generiert (bzw. eingegeben) werden können, ist die Anzahl an möglichen Notenhöhen $|N_{hö}| = 28$. Es werden außerdem nur Achtelnoten $|A| = 8$ als kleinstmöglicher Notenwert zugelassen.

Bei einer durchschnittlichen Länge von $|T| = 11$ Takten ergeben sich

$$|T| \cdot |A| \cdot |N_{hö}| \cdot |Stimmen| = 11 \cdot 8 \cdot 28 \cdot 2 = 4928 \text{ mögliche Notenpositionen}$$

und unter Berücksichtigung, dass pro Stimme nur eine Note zu jeder Zeit erklingen darf:

$$\left(|N_{hö}|^{|T| \cdot |A|}\right)^2 = 28^{176} \approx 5,01 \cdot 10^{254} \text{ mögliche Variationen.}$$

Für alle Mutationen wird ein *pseudo random number generator* basierend auf dem „Mersenne Twister“-Algorithmus [Matsumoto & Nishimura, 1998] verwendet.

4.1 Handgeschriebene Daten

Vorab wurden Kontrapunkte aus [Mann [Übersetzung] & Fux, 1965] eingegeben und als Trainingsdaten abgespeichert. Da im Buch nur 51 verschiedene Kontrapunkte zu finden sind, die sich als Trainingsdaten eignen, die alle auf nur 6 verschiedenen *cantus firmi* beruhen, wurden Versuche unternommen, aus diesen vom Menschen erstellten Daten weitere Daten zu mutieren.



Abb. 17 Von Hand geschriebener Kontrapunkt aus dem Gradus ad Parnassum (fig. 85).

4.2 Mutations-Kontrapunktgenerator

Da die handgeschriebenen Daten aus einem Lehrbuch [Mann [Übersetzung] & Fux, 1965] stammen und vermitteln sollen, wie man es richtig macht, enthalten sie nur wenige Noten, die die Kontrapunktregeln verletzen. Der Mutations-Kontrapunktgenerator erhält die handgeschriebenen Daten und wählt zufällig zwei Noten pro Stimme aus, die er zu neuen Noten mutiert.

Hat ein Kontrapunkt zum Beispiel 39 Noten in der oberen und 13 Noten in der unteren Stimme, werden zu Beginn der Mutation für die obere Stimme 2 Indices im Bereich 0–38 und für die untere im Bereich 0–12 generiert. Anschließend werden die entsprechenden Noten mutiert.

Bei der Mutation wird nur die Tonhöhe einer Note verändert. In jeweils $\frac{16}{17}$ der Fälle wird die Tonhöhe nach der Mutation eine andere sein, in $\frac{1}{17}$ der Fälle wird die Tonhöhe dieselbe bleiben. Zu um einen Halbton erhöhten bzw. verminderten Noten wird nicht mutiert.



Abb. 18 Ein aus fig. 88 aus dem *Gradus ad Parnassum mutierter Sheet* (sheet 0 der Mutationsvaliditätsdaten). Die umkreisten Noten wurden mutiert (Originalnoten schemenhaft abgebildet).

4.3 Stochastischer-Kontrapunktgenerator

Der stochastische Kontrapunktgenerator wendet einfache Regeln an, um zufällige Kontrapunkte zu generieren. Diese Kontrapunkte widersprechen größtenteils den Regeln nach Fux und klingen nicht sehr ansprechend. Es werden sowohl *c.f.* als auch *c.p.* generiert.

Zu Beginn wird jeweils mit einer 50-prozentigen Wahrscheinlichkeit festgelegt, ob der *c.f.* in der unteren oder der oberen Stimme steht. Anschließend wird der *c.f.* generiert.

4.3.1 *c.f.*

Für den *c.f.* werden nur ganze Noten und keine um einen Halbton erhöhten bzw. verminderten Noten erzeugt. Zusätzlich werden Regeln vorgegeben, um ihn möglichst wie einen realen *cantus firmus* generieren zu lassen. Die Position einer Note hängt dabei von ihrer Vorgängernote ab.

Die erste Note wird dabei mit einer Wahrscheinlichkeit von $\frac{1}{17}$ ausgewählt. Die nächsten Noten werden jeweils mit einer Wahrscheinlichkeit von $\frac{4}{5}$

einen kleinen Sprung und in $\frac{1}{5}$ der Fälle einen großen Sprung von der Vorgängernote entfernt platziert.

In jeweils $\frac{4}{5} \cdot \frac{1}{2} = \frac{4}{10}$ der Fälle wird das Sprungintervall 2 oder 3 betragen. In $\frac{1}{5}$

$\frac{1}{8} = \frac{1}{40}$ der Fälle wird das Intervall

zwischen einschließlich 2 und 9 liegen.

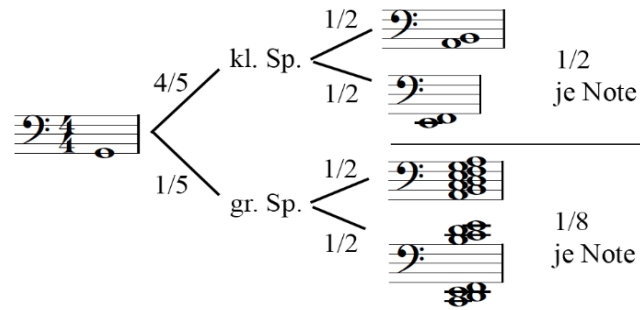


Abb. 19 Relative Wahrscheinlichkeiten einer generierten Folgenote. Links: Ausgangsnote, Rechts: mögliche Folgenoten.

In jeweils $\frac{1}{2}$ der Fälle wird der Sprung nach oben bzw. unten getätigt. Geht ein Sprung über die zugelassenen 17 Töne hinaus, wird er von der gegenüberliegenden Seite (oben/unten) fortgesetzt.

In $\frac{4}{5} \cdot 2 \cdot \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{5} \cdot 2 \cdot \frac{1}{2} \cdot \frac{1}{8} = \frac{17}{40}$ der Fälle ist das Sprungintervall $s = \pm 2$, in $\frac{4}{5} \cdot 2 \cdot \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{5} \cdot 2 \cdot \frac{1}{2} \cdot \frac{1}{8} = \frac{17}{40}$ ist es $s = \pm 3$ und in $\frac{1}{5} \cdot 2 \cdot \frac{1}{2} \cdot \frac{1}{8} = \frac{1}{40}$ der Fälle ist es $s = \pm x, x \in \{4, 5, 6, 7, 8, 9\}$.

4.3.2 c.p.

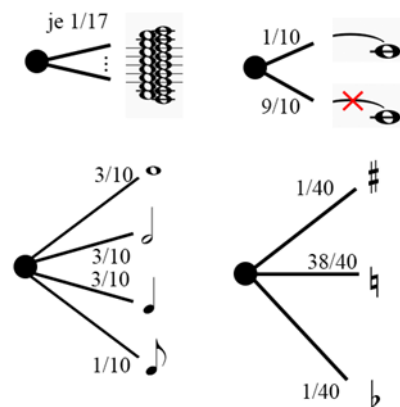
Für die Gegenstimme werden Noten unabhängig von ihrer Vorgängernote generiert. Es wird zuerst der Notenwert bestimmt. Jeweils zu $\frac{3}{10}$ wird dabei eine ganze, halbe oder Viertelnote generiert, in $\frac{1}{10}$ der Fälle eine Achtelnote. Diese Verteilung soll dazu beitragen, dass die neuronalen Netze nicht mit Achtelnoten überschwemmt werden, für die nur die Regel **R9** gilt.

Anschließend wird jeweils mit einer Chance von $\frac{1}{17}$ eine Notenhöhe ausgewählt. Diese Notenhöhe wird in $\frac{1}{40}$ der

Fälle um einen Halbton erhöht und in $\frac{1}{40}$ um einen

Halbton vermindert, in $\frac{38}{40}$ bleibt die Notenhöhe gleich.

Auch hiermit soll versucht werden, das Netz nicht mit unwichtigen Daten zu überfluten – Fux hat mit der Ausnahme von **R4** keine Regeln zu erhöhten oder verminderten Noten festgelegt.



In $\frac{1}{10}$ der Fälle wird die Note mit einem Haltebogen mit ihrer Vorgängernote verbunden, wird die erste Note im Takt ausgewählt, wird sie zu einer Pause. Werden Noten mit einem Wert eingefügt, der einen Taktstrich überschreitet, entstehen auch ohne diesen Schritt gehaltene Noten.

Abb. 20 Relative Wahrscheinlichkeiten der c.p.-Noten. Alle Eigenschaften sind voneinander unabhängig.

Da Achtelnoten laut Fux nur in Zweierpärchen auftreten sollen, wird diese Eigenschaft auch für den Generator übernommen:

Die Regeln nach Fux unterscheiden nur zwischen den Beat-Positionen *DBeat*, *UBeat* und den beiden *WBeats*. Würde eine einzelne Achtelnote eingefügt, würden sich alle Folgenoten auf einer undefinierten Beat-Position befinden und es könnte nicht mehr überprüft werden, welche Regeln verletzt werden.

Die zweite Achtelnote wird nach demselben Prinzip wie die anderen Noten generiert, nur dass der Notenwert (Achtel) feststeht.



Abb. 21 Ein typischer mit dem stochastischen Kontrapunktgenerator erstellter Kontrapunkt (sheet 0 der stochastischen Trainingsdaten).

4.4 Analyse der generierten Daten

Es wurden insgesamt drei verschiedene Datensätze, bestehend aus jeweils 10000 Trainings-, 2000 Test- und 2000 Validationskontrapunkten, erstellt. Die Trainingsdaten werden verwendet, um Netze zu trainieren, und anhand der Testdaten wird während des Trainings ein Vergleichs-Loss berechnet.

Anhand der Validationsdaten wird anschließend eine *Confusion Matrix* der Netze erstellt.

4.4.1 Postseparierte Mutationsdaten

Die postseparierten Mutationsdaten wurden mit dem Mutations-Kontrapunktgenerator aus allen 51 handgeschriebenen Kontrapunkten generiert und anschließend in Trainings-, Test- und Validationsdaten aufgeteilt.

Diese Daten sollen in erster Linie zum Testen der Netze verwendet werden.

4.4.2 Präseparierte Mutationsdaten

Die präseparierten Mutationsdaten wurden ebenfalls mit dem Mutations-Kontrapunktgenerator generiert, allerdings wurden hierfür zuerst zufallsgesteuert die 51 handgeschriebenen Kontrapunkte in ein Trainingsset von 37 und ein Test-/Validationsset¹⁷ von 14 Kontrapunkten aufgeteilt. Durch das Unterteilen der Test- und Trainingsdaten vor dem Generieren soll verhindert werden, dass zu viele gleiche Datensätze entstehen.

Anschließend wurden mit dem Mutations-Kontrapunktgenerator die Daten generiert und diese im Fall der Test-/Validationsdaten aufgeteilt.

¹⁷ Wegen den wenigen zu Verfügung stehenden Daten werden die Test- und Validationsdaten anhand derselben Ausgangskontrapunkte generiert.

Anhand dieser Daten soll geprüft werden, ob ein Lernerfolg auch mit vielen sehr ähnlichen Beispielen möglich ist, wie sie auch in der Praxis vorliegen würden, wenn von Menschenhand erstellte Kontrapunkte verwendet würden.

4.4.3 Stochastische Daten

Die stochastischen Daten wurden mit dem stochastischen Kontrapunktgenerator generiert und anschließend in Trainings-, Test- und Validationsdaten aufgeteilt.

Diese Daten sollen verwendet werden, um die Netze zu trainieren.

4.4.4 Vergleich der Daten

Ein Lernerfolg oder -misserfolg der neuronalen Netze könnte durch fehlerhafte oder sich ähnelnde Datenmengen zustande kommen. Da die Daten mit Vorgaben durch den Computer generiert wurden, ist diese Gefahr umso größer und sie soll durch die folgende Analyse so gut wie möglich verringert werden.

In Abb. 22 ist ersichtlich, dass sich die stochastischen Daten in der Verteilung der vertretenen Notenhöhe sehr von den Mutationsdaten unterscheiden. In den mutierten Daten treten viele Noten im Bereich D3 bis F3 und D4 bis G4 auf. Das sind genau die Bereiche, in denen die ursprünglichen Daten auch am häufigsten vertreten sind: die mittlere Tonhöhe in der jeweiligen Stimme. Bei den stochastischen Daten sind alle Notenhöhen gleichverteilt und in den Noten, die jeweils zur oberen oder zur unteren Stimme gehören können, doppelt so hoch. Für die Verteilung der *c.p.*-Notenhöhen sieht das Bild ähnlich aus (Anhang: „Vollständige Analyse der generierten Daten“ S. 60).

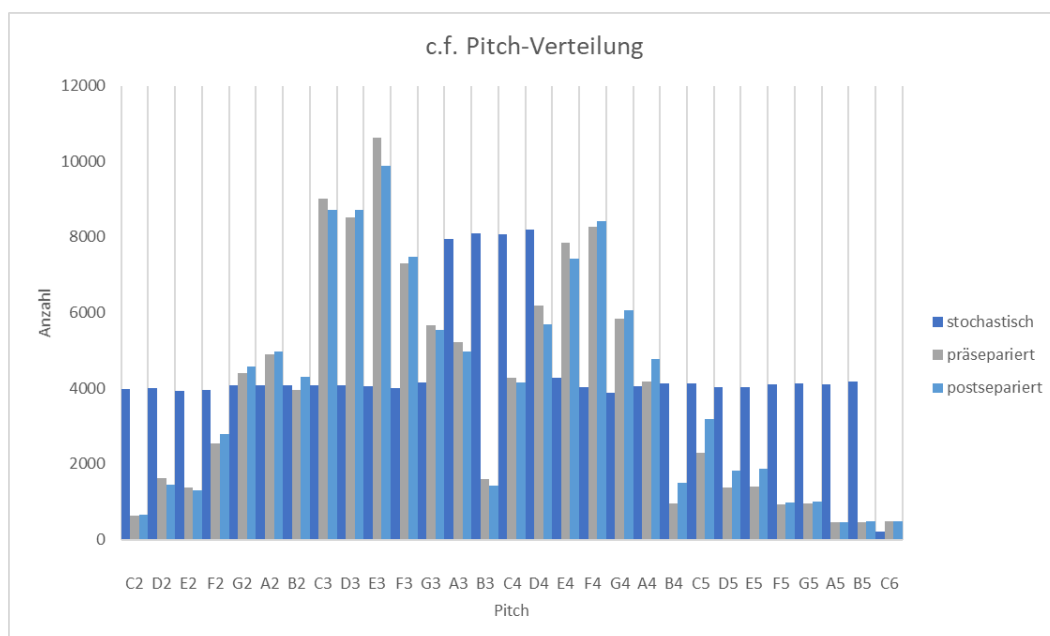


Abb. 22 Vertretene Notenhöhen (Pitch) im cantus firmus der verschiedenen Daten. Aufgrund eines Programmierfehlers wird C6 nur erzeugt, wenn es durch einen Sprung, der über das untere Ende der oberen Stimme hinausgeht, generiert wird. Dieser Fehler wurde erst erkannt, nachdem schon viele Versuche mit den Netzen abgeschlossen waren, und konnte aus Zeitgründen nicht berichtigt werden. Er sollte aber keinen Einfluss auf den Lernerfolg der Netze haben.

Die Notenwerte der *c.p.*-Noten Abb. 23 sehen in stochastischen und mutierten Daten ähnlich aus, nur die Achtelnoten kommen in den stochastischen Daten häufiger vor. Wie man aus den gebrochenen Regeln in Abb. 25 ablesen kann, ist die Anzahl an gebrochenen Regeln R9 (bezieht sich auf Achtelnoten) in den stochastischen Daten im Vergleich sehr hoch. Es könnte sich also positiv auf das Lernen dieser Regel auswirken

Für die in Abb. 24 dargestellte Bewegung (*Motion*) der Noten beinhalten die stochastischen Daten mehr *Oblique Motion* als die mutierten. In den Versuchen, die mit den Netzen durchgeführt wurden, schien dies für die Erkennung der Regeln, die mit Bewegung zu tun haben, allerdings keine Rolle zu spielen.

Die in Abb. 25 dargestellten gebrochenen Regeln unterscheiden sich in ihrer Anzahl erheblich. Regeln wie R1, die sich nur auf den Anfang und das Ende des Kontrapunkts beziehen, sind erwartungsgemäß seltener vertreten.

Regel R7 „Gehaltene Noten nur auf dem *downbeat*“ wird in den mutierten Daten gar nicht verletzt und kann aus diesem Grund natürlich auch nicht

gelernt werden, wenn diese Daten zum Trainieren der Netze verwendet werden. Umgekehrt kann ein Netz, das auf den stochastischen Daten trainiert wurde, dennoch auf die *False-Positives*¹⁸ in Bezug auf die mutierten Daten untersucht werden.¹⁹

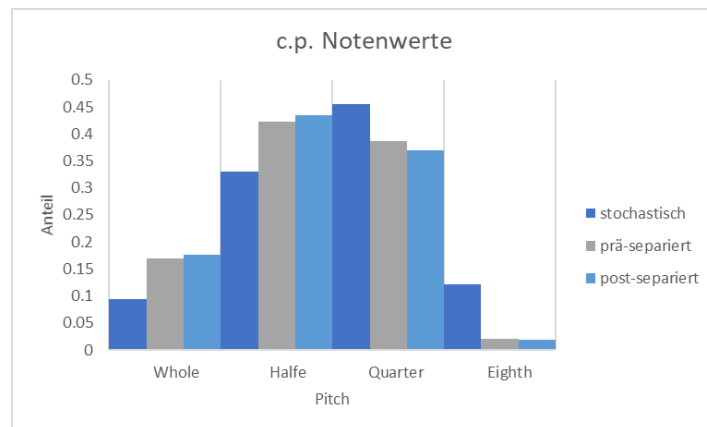


Abb. 23 Verteilung der Notenwerte im *c.p.*

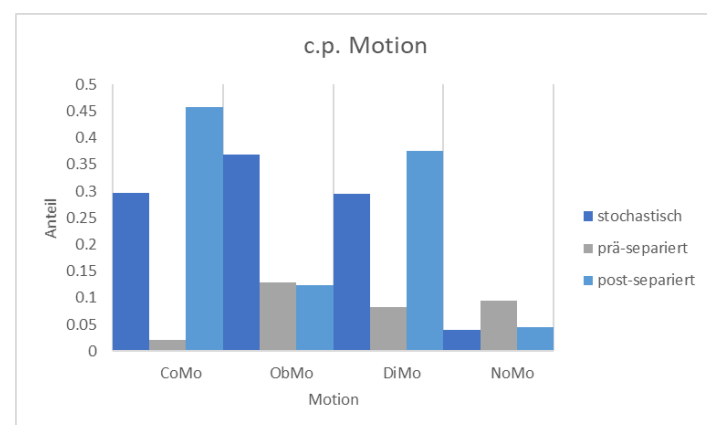


Abb. 24 Erzeugte Bewegung der Noten. *NoMo* bezieht sich immer auf die erste Note, da zu ihr keine Bewegung stattfindet.

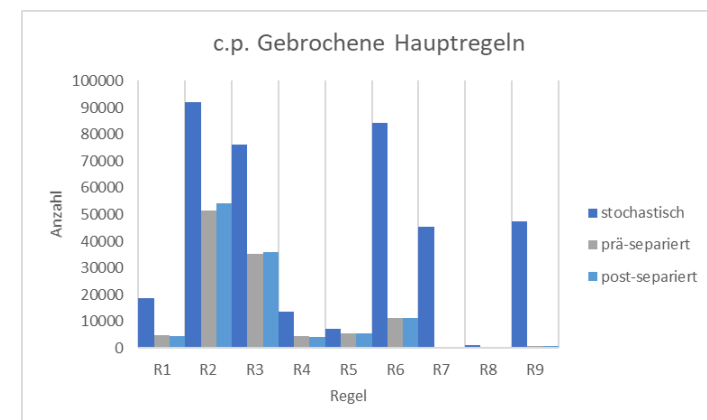


Abb. 25 Die gebrochenen Kontrapunktregeln.

¹⁸ Die Inputs, die das Netz fälschlicherweise für wahr (Regel ist gebrochen) hält.

¹⁹ Bei den Versuchen wurde allerdings nur ein einziges Mal eine R7 fälschlicherweise als gebrochen bewertet. Aus diesem Grund wird R7 nicht für die Auswertung der Netze beachtet.

Für R8 „Verbotene *Suspensions*“ liegen nur wenige Elemente vor, die die Regel brechen. In den stochastischen Daten 1.141 von 335.006 Elementen, in den prämutierten Daten 124 von 235.628 Elementen und in den postmutierten Daten 286 von 235.953 Elementen.²⁰ R8 stellt damit einen Extremfall dar, der genauer untersucht werden soll. Da diese Regel nur von der betrachteten und der nachfolgenden Note abhängt, ist besonders darauf zu achten, ob nur ein einfaches Auswendiglernen stattfindet.

²⁰ Im Text werden Tausendertrennzeichen verwendet, um ein flüssiges Lesen zu ermöglichen. In mathematischen Formeln und Tabellen wird darauf verzichtet.

4.5 Repräsentation der Noten

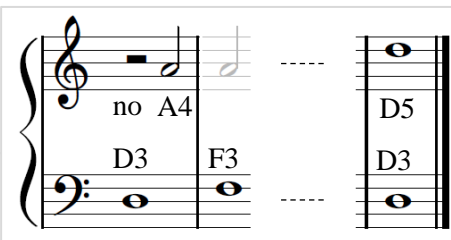
Die so generierten Daten liegen im sheet-Format vor (Abb. 26). Sie werden in ein binäres Format umgewandelt, damit sie von den neuronalen Netzen verarbeitet werden können.

Um zu bestimmen, ob die Fux-Regeln eingehalten werden oder nicht, werden die umliegenden Noten einer betrachteten Note benötigt. Welche Noten für die Bewertung aber wirklich notwendig sind, unterscheidet sich je nachdem, welche Noten vorhanden sind.

Soll zum Beispiel ein, wie in Abb. 27 dargestelltes, gehaltenes **C** bewertet werden, muss geprüft werden, ob es richtig aufgelöst wird. Das nachfolgende **H** und die Note **D** des *c.f.* spielen eine Rolle. Wie schon im vorherigen Kapitel beschrieben, müssen nur Achtelnoten betrachtet werden, da für kleinere Notenwerte keine Regeln existieren. Es ergäbe sich also ein Bereich von 3 Achtelnoten, um diese eine Regel bewerten zu können. (vgl. Abb. 29)

In Abb. 28 ist eine einfachere Melodie gegen den *c.f.* gesetzt. Für die Bewertung der Note **H** spielen jetzt die *c.p.*-Note **H** und die *c.f.*-Note **E** des vorherigen sowie die *c.f.*-Note **H** des betrachteten Takts eine Rolle. Als Achtelnoten müssten 9 Scheiben betrachtet werden.

Um jede Regel nach Fux richtig bewerten zu können, müssen immer mindestens der gesamte letzte und aktuelle Takt, sowie die erste Note des Folgetakts des *c.f.* und *c.p.* betrachtet werden.



tempo:
200
cf:
bass
soprano:
no p: no va: 2 vo: 1 f: 0 s: 0 t: 1 | {}
A4 p: 69 va: 2 vo: 1 f: 0 s: 0 t: 0 | {}
[:]
D5 p: 74 va: 1 vo: 1 f: 0 s: 0 t: 0 | {}
bass:
D3 p: 50 va: 1 vo: 0 f: 0 s: 0 t: 0 | {}
F3 p: 53 va: 1 vo: 0 f: 0 s: 0 t: 0 | {}
[:]
D3 p: 50 va: 1 vo: 0 f: 0 s: 0 t: 0 | {}
end

Abb. 26 Das sheet-Format. Die Noten sind nacheinander aufgelistet. Die Zusätze sind wie folgt definiert: p: 69 ist die Tonhöhe (pitch) A4 im midi-Format, va: 2 (Notenwert: value) gibt an, dass es sich um eine halbe Note handelt, f: 0, s: 0 bedeutet, dass die Note weder vermindert (flat) noch erhöht (sharp) ist, t: 0 bedeutet, dass die Note nicht gehalten (tied) ist.

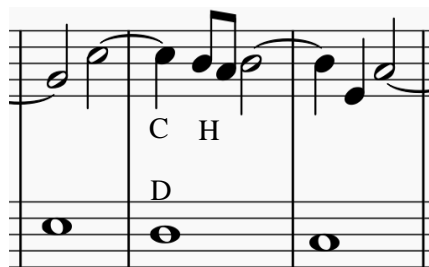


Abb. 27 Regeln des 5. Sp. Counterpoint

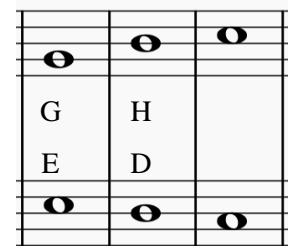


Abb. 28 Regeln des 1. Sp. Counterpoint

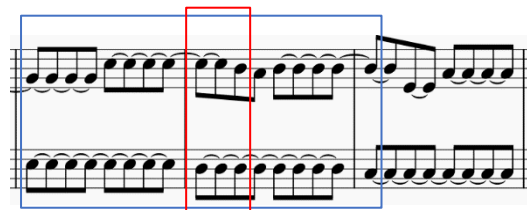
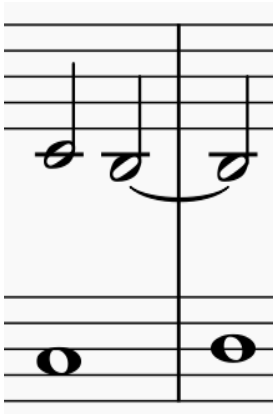


Abb. 29 Die Noten aus Abb. 27 dargestellt als Achtelnoten. Blau umrandet der Bereich, der zur Bewertung aller Kontrapunktregeln benötigt wird. Rot umrandet der Bereich, der benötigt wird, um bewerten zu können, ob das C richtig aufgelöst wurde.

Dazu werden die Noten in Achtelnoten umgewandelt (Abb. 29) und in eine Matrix konvertiert (Abb. 30). Die Tonhöhe einer Note wird durch eine 1 an der entsprechenden Stelle gekennzeichnet.

Da Halbtöne im Kontrapunkt nur selten auftauchen, werden ausschließlich Noten ohne Vorzeichen abgebildet, und für erhöhte (*sharp*) oder verminderte (*flat*) Noten werden zwei weitere Zeilen in die Matrix eingefügt, die entsprechend gesetzt werden. Für den *c.f.* kann auf diese zusätzlichen Stellen verzichtet werden, da im Kontrapunkt nach Fux der *c.f.* nur Töne ohne Vorzeichen enthalten darf.



C6	00000000 0000 ...
:	
E4	00000000 0000 ...
D4	00000000 0000 ...
C4	11110000 0000 ...
B3	00001111 1111 ...
A3	00000000 0000 ...
sharp	00000000 0000 ...
flat	00000000 0000 ...
tied	00000000 1111 ...
new note	10001000 0000 ...
E4	00000000 0000 ...
:	
F3	00000000 0000 ...
E3	00000000 0000 ...
D3	00000000 1111 ...
C3	11111111 0000 ...
B2	00000000 0000 ...
:	
C2	00000000 0000 ...
new note	10000000 1000 ...

Abb. 30 Ausschnitt der binären Codierung. Die Taktstriche sind nur zur Orientierung eingetragen und nicht Teil des Codes.

Beide Stimmen erhalten ihre eigenen Zeilen, die die Tonhöhe angeben. Für die Ganztonschritte in der Sopranostimme gilt der Bereich A3 bis C6, im Bass der Bereich C2 bis E4. Die Noten A3, B3, C4, D4 und E4 tauchen also insgesamt zweimal auf.²¹

Eine Note *c.p.*, die gehalten (*tied*) ist, wird mit einer weiteren 1 gekennzeichnet.

Eine neue Note wird an ihrem Beginn durch die zusätzliche Information „new note“ gekennzeichnet.²²

Eine einzelne Achtelnotenscheibe besteht bei dieser Codierung aus 39, ein Takt aus 312 Elementen.

Das reduziert die am Anfang des Kapitels beschriebenen Notenpositionen von 4.928 auf

$$|T| \cdot |Code| = 11 \cdot 312 = 3432 \text{ mögliche Notenpositionen.}$$

Für die Berechnung der möglichen Variationen ergibt sich für den *c.f.* eine reduzierte Anzahl der Notenhöhen $|N_{hö_{c.f.}}| = 17$. Weiter wird die Anzahl der möglichen *cantus firmi* reduziert, da hier nur ganze Noten, also eine Note pro Takt, generiert werden. Die Anzahl der möglichen *c.p.* Variationen bleibt gleich und die Anzahl insgesamt sinkt von $5,01 \cdot 10^{254}$ auf

$$|N_{hö_{c.f.}}|^{|T|} \cdot |N_{hö_{c.p.}}|^{|T| \cdot |A|} = 17^{11} \cdot 28^{11 \cdot 8} \approx 7,67 \cdot 10^{140} \text{ mögliche Variationen.}$$

²¹ Auf einem Klavier könnten diese Noten zwar nicht gleichzeitig gespielt werden, in einem Chor ist es aber möglich, dass zwei Stimmen dieselbe Note singen. Das Intervall wäre hier 1 und die Note würde zu beiden Melodien gehören.

²² Da im zweistimmigen Kontrapunkt pro Stimme nur eine Note zu jeder Zeil erklingen kann, reichen diese beiden Informationen aus, um jede Note exakt zu bestimmen.

4.5.1 Gleiche Datensätze

Um herauszufinden, ob die neuronalen Netze einfach den Input der Trainingsdaten auswendig lernen könnten, sollen die Datensätze auf gleiche Elemente untersucht werden.²³

Wie diese Elemente genau aussehen, wird auf der nächsten Seite beschrieben. Zunächst wird erläutert, wie die gleichen Datensätze ermittelt werden (vgl. auch Abb. 31) und welcher Vorteil daraus entsteht:

Es werden die n-Tupel a und b (z.B. a = stochastische Trainingsdaten und b = postseparierte Mutationstestdaten) als Mengen A und B abgebildet und anschließend die in beiden Mengen vorkommenden Elemente D_u bestimmt:

$$D_u = A \cap B$$

Aus der Menge dieser Elemente und den n-Tupeln a und b können dann die n-Tupel d_a und d_b gebildet werden. Diese enthalten jeweils alle doppelten Elemente so oft, wie sie in den n-Tupeln a und b vorkommen:

$$d_a = (\forall x \in a | x \in D_u)$$

$$d_b = (\forall x \in b | x \in D_u)$$

Für alle Elemente kann außerdem festgestellt werden, ob sie eine Regel brechen. Dies wird im Folgenden durch die hochgestellte Information R1 bis R9 gekennzeichnet: Das n-Tupel d_a^{R8} würde also alle Elemente, die Regel R8 brechen, so oft enthalten, wie sie in a vorkommen.

Würde Regel R8 in den Validationsdaten zum Beispiel 100-mal gebrochen und das Netz konnte 73-mal erkennen, dass die Regel gebrochen wurde, wäre das – dem Anschein nach – ein zufriedenstellendes Ergebnis. Läge für die betrachteten Daten allerdings die Zusatzinformation $|d_b^{R8}| = 73$, mit a = Trainingsdaten (enthalten in d_b) und b = Validationsdaten vor, wäre anzunehmen, dass das Netz nur Elemente erkennen konnte, die es bereits exakt so in den Trainingsdaten gesehen hat.

$$\begin{aligned} a &= (4, 3, 9, 1_{R3}, 4, 8_{R3}) \\ b &= (1_{R3}, 4, 2, 7, 1_{R3}, 4, 8_{R3}) \\ A &= \{1, 3, 4, 9, 12\} \\ B &= \{1, 2, 4, 7, 8\} \\ D_u &= \{1, 4, 8\} \\ d_a &= (1_{R3}, 4, 4, 8_{R3}) \\ |d_a| &= 4 \\ |d_a^{R3}| &= 2 \\ d_b &= (1_{R3}, 4, 1_{R3}, 4, 8_{R3}) \\ |d_b^{R3}| &= 3 \end{aligned}$$

Abb. 31 Die Elemente werden hier vereinfacht durch ganze Zahlen dargestellt. In diesem Beispiel wird von einigen Elementen Regel R3 verletzt (In den n-Tupeln durch das tiefgestellte R3 angedeutet).

²³ Werden die Netze mit den postseparierten Daten trainiert und validiert, können die Netze für einige Regeln fast eine hundertprozentige Trefferquote erreichen. (vgl. Versuchsreihe „Training mit postseparierten Mutationsdaten“ S. 44) Um auszuschließen, dass diese Ergebnisse nur aufgrund doppelter Elemente in Trainings- und Validationsdaten zustande kommen, wurden die in diesem Kapitel beschriebenen Vergleiche unternommen.

Um die Elemente zu bilden, werden die Notenblätter in die vorgestellte binäre Form gebracht und in Bereiche aufgeteilt, die für die Kontrapunktregeln entscheidend sind.

Für die meisten Kontrapunktregeln ist der Takt, in dem sich die *c.p.*-Note befindet (für die die Regeln geprüft werden sollen), der Takt davor und der Takt danach interessant.²⁴

Um auf Ähnlichkeit zu testen, wurden jeweils alle Test-, Trainings- und Validationsdaten in Segmente aus jeweils 17 Achtelnotenscheiben unterteilt und anschließend verglichen.

Eine so codierte Matrix (Abb. 32) besteht aus

$$(2 \cdot 8 + 1) \cdot 39 = 663 \text{ Elementen.}^{25}$$

Da für jede *c.p.*-Note eines Kontrapunkts eine solche Matrix erzeugt wird, hängt die Anzahl der Matrizen direkt von der Anzahl der *c.p.*-Noten ab. Für die 10.000 mit dem stochastischen Kontrapunktgenerator erzeugten Trainingsdaten ergeben sich so 335.006 Matrizen.

Sollen diese nun zum Beispiel mit den 46.883 Matrizen der postseparierten Mutations-Testdaten verglichen werden, müssen

$(335006 \cdot 663) \cdot (46883 \cdot 663) \approx 6,90 \cdot 10^{15}$ Vergleiche stattfinden.²⁶

Um diesen Vorgang zu beschleunigen, wurden drei alternative Codierungen der Matrizen untersucht.

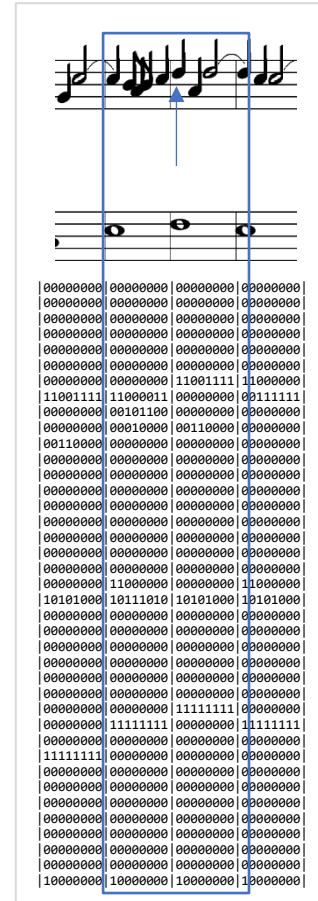


Abb. 32 Ausschnitt eines Kontrapunkts, dargestellt als Notenblatt und formatierter Code. Die Matrix, die für die Prüfung auf Ähnlichkeit generiert würde, ist blau umrandet und bezieht sich auf die mit dem Pfeil versehene Note.

²⁴ Für einige Regeln ist auch schon ein kleinerer Bereich ausreichend, darum wird später noch beschrieben, welche Bereiche genau betrachtet werden. Zunächst das Beispiel anhand des großen Bereichs.

²⁵ 8 Achtelnotenscheiben links, 8 rechts plus die Scheibe, in der sich die betrachtete Note befindet.

²⁶ Bei einem Versuch ergab sich beim einfachen Vergleichen Zahl gegen Zahl eine Testdauer von ungefähr 4 Stunden.

Pair-Code

Als „Pair-Code“ wird hier ein Code bezeichnet, bei dem eine Zahlenreihe als Pärchen aus dem Zeichen eines Alphabets und der Anzahl dieser Zeichen beschrieben wird.

Da das Alphabet $A = \{0, 1\}$ sehr klein ist und die Matrizen oft lange Reihen desselben Zeichens enthalten, kann die Codewortlänge erheblich reduziert werden. In (Abb. 33) von 663 Elementen auf 49 Pärchen, also 98 Elemente, die verglichen werden müssen.

Mit dieser Codierung können die Matrizen der stochastischen Trainingsdaten im Schnitt durch 104,10 Elemente abgebildet werden, die der postseparierten Mutationsdaten durch 100,33 Elemente. Im ungünstigsten Fall ergeben sich

Mitte: Pair-Code mit 49 Pärchen. Unten: 64bit-Code mit 11 64bit-Integern.

$$(335006 \cdot 104,101) \cdot (46883 \cdot 100,327) \approx 1,64 \cdot 10^{14} \text{ Vergleiche.}$$

Ein Vergleich kann allerdings schon beim Vergleich der ersten Pärchen abgebrochen werden, wenn nur die erste 0 oder 1 auf einer anderen Position ist, dadurch ergibt sich eine untere Grenze von

$$(335006) \cdot (46883) \approx 1,57 \cdot 10^{10} \text{ Vergleichen.}$$

64bit-Codierung

Bei der 64bit-Codierung werden die Elemente der Matrizen in 64bit-Integer umgewandelt. Die

663 Elemente können so als nur $\left\lceil \frac{663}{64} \right\rceil = 11$ Werte dargestellt werden.

Es ergibt sich eine obere Schranke von

$$(335006 \cdot 11) \cdot (46883 \cdot 11) \approx 1,90 \cdot 10^{12} \text{ Vergleichen.}$$

704bit-Codierung

Werden die elf 64bit-Werte nicht einzeln, sondern als eine einzige 704bit-Zahl betrachtet, ist für jedes Element $e_n \in A$ definiert, ob $e_1 < e_2$. So können algorithmische Verfahren verwendet werden, um die mehrfach vorhandenen Elemente zu identifizieren.

Wird auf die n -Tupel der zu vergleichenden Elemente eine *Mergesort* angewendet, kann in Folge eine *Binary-Search* die mehrfach vorkommenden Elemente finden. Es ergibt sich, dass nicht mehr jedes Element mit jedem Element des anderen n -Tupels verglichen werden muss, und die Komplexität kann aus dem $O(n^2)$ -Raum in den $O(n \cdot \log(n))$ -Raum überführt werden. (Abb. 34)

Originalmatrix

[illegible]

Pair-Code

(325, 0)(1, 1)(14, 0)(2, 1)(9, 0)(1, 1)(7, 0)(1, 1)(5, 0)(1, 1)(14, 0)(1, 1)(10, 0)(1, 1)(13, 0)(1, 1)(14, 0)(1, 1)(10, 0)(1, 1)(13, 0)(1, 1)(14, 0)(1, 1)(10, 0)(1, 1)(9, 0)(1, 1)(13, 0)(1, 1)(25, 0)(1, 1)(13, 0)(1, 1)(25, 0)(1, 1)(13, 0)(1, 1)(14, 0)(2, 1)(2, 0)

64bit-Code

0, 0, 0, 0, 0, 576487149206503440, 144123984437444616,
4505798784845826, 2251799847241728, 562984313160192, 262156

Abb. 33 Oben: Originalmatrix mit 663 Elementen. Mitte: Pair-Code mit 49 Pärchen. Unten: 64bit-Code mit 11 64bit-Integern.

pair	64bit	704bit
Dauer	Dauer	Dauer
01:24:36	02:04:41	00:04:34

Abb. 34 Ein Performance-Test, bei dem die postseparierten Trainingsdaten mit den Test- und Validationsdaten verglichen wurden, zeigt den Vorteil der 704bit-Lösung. Verglichen mit der 64bit-codierten Lösung entsteht eine Zeitersparnis von 96,34%.

Vergleich der Daten

Da für manche Regeln nur die vorherige und die nachfolgende Note eine Rolle spielen, wurden zusätzlich zu dem beschriebenen Elementbereich zwei weitere, kleinere Bereiche um die Note untersucht: einer, der in beide Richtungen 4 Achtelscheiben (insgesamt 9) und einer, der 2 Achtelscheiben (insgesamt 5) betrachtet. (Abb. 35)

Es werden, außer dem in Abb. 35 dargestellten Bereich, noch 9 Bit angehängt, die angeben, welche Regeln gebrochen wurden. Ohne diese Angabe würde der Vergleich auch Elemente als mehrfach vorhanden ansehen, die keinen negativen Einfluss auf die Bewertung der Netze hätten.

In Abb. 36 sind die $|d_b^{Rx}|$ -Werte für die 17 Achtelnotenscheiben umfassenden Bereiche aufgelistet. Das a ist dabei als die jeweiligen Trainingsdaten definiert und b als die Test- bzw. Validationsdaten.

Es ist zu erkennen, dass sich die stochastischen Trainingsdaten sehr von den postseparierten Daten unterscheiden. Sogar für R2, die in den stochastischen Trainingsdaten 92.174-mal und in den postseparierten Validationsdaten 10.889-mal gebrochen wird, sind nur 26 Elemente in beiden Datensätzen vorhanden.

Alarmierend sind die Ergebnisse der postseparierten Validationsdaten. Regel R8 wird hier insgesamt nur 52-mal gebrochen, die 30 mit den Trainingsdaten übereinstimmenden Datensätze machen also rund 60% aus.

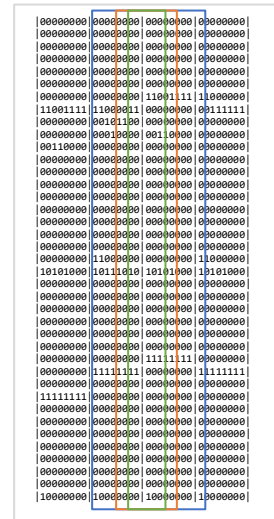


Abb. 35 Die Daten aus Abb. 32, blau umrandet die Matrix für 17 Achtelscheiben, orange für 9 und grün für 5.

$ d_b^{Rx} $ für Elemente aus 17 Achtelnotenscheiben		R1	R2	R3	R4	R5	R6	R7	R8	R9
a	b									
stoch train	post test	17	25	1	0	7	9	0	0	0
	post valid	12	26	2	0	6	4	0	0	0
post train	post test	502	6438	3894	352	551	578	0	41	17
	post valid	480	6454	3875	313	548	558	0	30	13
prä train	prä test	145	1600	1070	88	197	181	0	1	0
	prä valid	154	1586	1074	98	180	146	0	2	0

Abb. 36 stoch: stochastische Daten, post: postseparierte Mutationsdaten, prä: präseparierte Mutationsdaten, train: Trainingsdaten, test: Testdaten, valid: Validationsdaten

5 Verwendete Netze

Dem Titel dieser Arbeit entsprechend sollen in erster Linie LSTMs²⁷ untersucht werden. Um Vergleichswerte zur Hand zu haben, wird zusätzlich ein klassisches Feedforward-Netz untersucht, das nur aus *Dense Layern* besteht.

5.1 Das Feedforward-Netz

In einem *Feedforward Neuronal Network* sind die Layer hintereinander angeordnet. Beim Trainieren eines solchen Netzes werden Informationen in den Input-Layer eingespeist, und die Werte werden über die Gewichte, Aktivierungsfunktionen und Biases der Hidden-Layer manipuliert und führen am Output-Layer zu einem oder mehreren Werten. Am Output-Layer können dann diese Werte mit den angestrebten Target-Values verglichen und der Loss berechnet werden. Damit das Netz lernt, findet anschließend die Backpropagation statt, bei der versucht wird, alle Werte im Netz so anzupassen, dass der Loss minimiert wird.

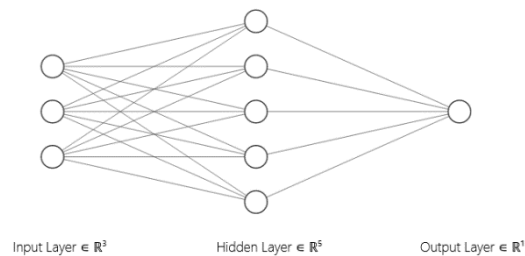


Abb. 37 Feedforward Netz mit einem Hidden Layer.

Bei der Backpropagation wird der Gradient berechnet, in dessen Richtung sich das nächste lokale Minimum befindet und das Netz in diese Richtung optimiert. Die Gradienten im Output-Layer können einfach über die erste Ableitung der Loss-Funktion ermittelt werden. Für jeden früheren Layer hängt der Gradient von der Ableitung des Folge-Layers ab.

Wird für die Berechnung des Loss eine Sigmoidfunktion wie $\sigma(x) = \frac{1}{(1+e^x)}$ verwendet, kann dabei das *vanishing gradient problem* auftreten (Abb. 38). Vor allem in frühen Layern liegen dann nur noch sehr kleine Werte zur Optimierung vor, und das Netz kann nicht trainiert werden.

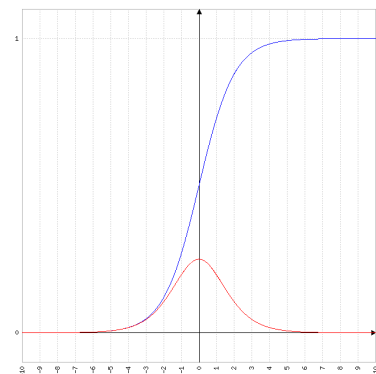


Abb. 38 $\sigma(x)$ (blau) und $\sigma'(x)$ (rot). Bei großen Abweichungen sind die Gradienten am kleinsten.

Sollen Input-Sequenzen gelernt werden, bei denen jeder spätere Input der Sequenz von den vorherigen abhängt, kann das Netz die komplette Sequenz als Input erhalten. Da jedes zusätzliche Inputneuron allerdings mit der gesamten ersten Hidden-Layer-Schicht verbunden ist, steigt die Zahl der Kanten in dem am stärksten vom Vanishing-gradient-Problem betroffenen Layer linear an. (vgl. Abb. 37, Abb. 40)

²⁷ Long-Short-Term-Memory-Netze

5.2 Das Long-Short-Term-Memory-Netz

Das LSTM, erstmals beschrieben in [Hochreiter & Schmidhuber, 1997], wird verwendet, da es eine Eigenschaft besitzt, das *vanishing gradient problem* zu umgehen und sich besonders zum Lernen von Sequenzen eignet.²⁸

Das Long-Short-Term-Memory (Abb. 39) führt einen internen *state* der Hidden-Layer ein. Dieser als Memory-Cell bezeichnete *state* besteht immer aus den vorherigen Inputs.

Das in dieser Arbeit verwendete LSTM orientiert sich an dem durch [Gers, Schmidhuber, & Cummins, 1999] beschriebenen LSTM mit *forget gate*.

Ein LSTM dieser Art besteht aus der Memory-Cell, in der der State (Abb. 39, 1) der Hidden-Layer gespeichert wird. Dem *forget gate* (Abb. 39, 2), das anhand des Outputs des aktuellen und des letzten Sequenzelements entscheidet, was aus der Memory-Cell vergessen werden kann, und das meist durch $\sigma(x) = \frac{1}{(1+e^x)}$ realisiert wird.

Nachdem das LSTM so entschieden hat, was vergessen werden soll, wird entschieden, was für neue Informationen gespeichert werden sollen (Abb. 39, 3). Dafür wird über $\tanh(x)$ berechnet, ob ein positiver oder negativer Wert zur Memory-Cell addiert werden soll, und mit $\sigma(x) = \frac{1}{(1+e^x)}$ das Ausmaß der Änderung bestimmt.

Abschließend wird im Output-Gate (Abb. 39, 4) mit dem Memory-Cell-State, dem aktuellen Input der Sequenz und dem Output des vorherigen Sequenzelements, der neue Output berechnet. Bei dieser Berechnung spielt der Input also nur noch für das Maß des Outputs eine Rolle, während der Memory-Cell-State auch noch das Vorzeichen bestimmen kann. Der so berechnete Wert ist dann der Output des Netzes für das aktuelle Sequenzelement.

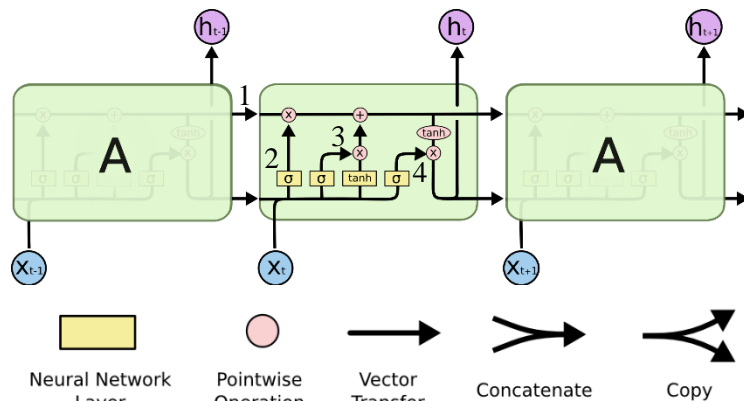


Abb. 39 Schematische Darstellung eines LSTM. x_t beschreibt den Input einer Sequenz an der Position t (zum Beispiel eine Achtelnote). h_t beschreibt den jeweiligen Output zur Sequenz an der Position t (die Netze in dieser Arbeit verwerfen alle Outputs außer den letzten).

1: State der Memory-Cell, 2: Forget Gate, 3: Speichern neuer Informationen, 4: Output-Gate.

Quelle: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (03.12.2019) (Nummerierung nachträglich eingefügt)

²⁸ Die hier beschriebene Erklärung orientiert sich an dem Blogpost [Olah, 2015].

Bei der Optimierung werden die Gewichte aller Gates angepasst. Es kann also theoretisch gelernt werden, was vergessen werden soll, welche Informationen des aktuellen Inputs und des letzten Outputs gespeichert werden sollen und schließlich wie der Memory-Cell-State den Output beeinflussen soll.

Durch diese verschiedenen optimierbaren Gewichte entsteht zunächst ein Overhead gegenüber dem Feedforward-Netz, es werden aber ungeachtet der Länge der Inputsequenz keine zusätzlichen Kanten benötigt. (Abb. 40)²⁹



Abb. 40 Anzahl der Kanten zwischen Input und Hidden Layer mit 312 Neuronen.
Feedforward (blau), LSTM (rot)
y: Anzahl Kanten, x: Betrachtete Achtennotenscheiben.

5.3 Dropout

Zusätzlich zum LSTM wird ein wie in [Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014] beschriebener

Dropout verwendet, um ein *Overfitting* des LSTM zu verhindern. Beim *Dropout* werden während des Lernprozesses zufällig Neuronen ausgewählt und deaktiviert. Für die Berechnung der Werte wird dabei so agiert, als ob die Neuronen nicht existieren. Dadurch, dass das Netz sich so nicht auf einzelne Neuronen verlassen kann, soll verhindert werden, dass ein einfaches Auswendiglernen der Daten stattfindet.

5.4 Loss-Functions

Für die Klassifizierung wird ein Binary Cross-Entropy-Loss³⁰ verwendet. Er eignet sich zur Klassifizierung von Elementen, die mehreren Klassen angehören können, und ist definiert durch:

$$l(x, y) = -\frac{1}{N} \sum_{n=1}^N (y_n \cdot \log(x_n) + (1 - y_n) \cdot \log(1 - x_n))$$

x : output, y : target, N : batchsize

5.5 Optimizer

Es wird der Adam-Optimizer nach [Kingma & Ba, 2015] verwendet. Adam setzt gegenüber dem klassischen SGD³¹ unterschiedliche Lernraten für die verschiedenen Gewichte des Netzes ein. Er besitzt eine Eigenschaft, die von den Autoren als „a form of automatic annealing“ bezeichnet wird und die bewirkt, dass die einzelnen Parameter weniger stark verändert werden, wenn sie dem Optimum näher kommen.

²⁹ Der Overhead in LSTM-Netzen mit mehreren Hidden-Layern kann allerdings so groß sein, dass erst bei einer erheblichen Sequenzlänge weniger lernbare Parameter als beim Feedforward-Netz vorhanden sind (vgl. Diskussion der Ergebnisse S. 46).

³⁰ Teils wird in der Literatur auch der Begriff Sigmoid Cross-Entropy Loss verwendet

³¹ Stochastic Gradient Descent

6 Bewertung der Kontrapunktregeln

Da es für die meisten Kontrapunktregeln egal ist, ob der **c.f.** in der im Soprano oder im Bass steht, wird der **c.p.** für den Input immer nach oben und der **c.f.** nach unten geschrieben. Um eine Unterscheidung zu ermöglichen, wird ein zusätzliches Feature hinzugefügt, das angibt, ob der **c.f.** im Bass steht. So ergibt sich für eine Achtelnotenscheibe eine Gesamtanzahl von 40 Features.

6.1 Gewählte Inputs

Alle Inputs basieren auf den Achtelnotenscheiben (Abb. 42), wie sie unter „Repräsentation der Noten“, S.22 beschrieben wurden, plus dem Feature das angibt, ob sich der *c.f.* im Bass befindet. Insgesamt wurden drei verschiedene Inputs untersucht. Davon zwei für das LSTM und einer für das Feedforward-Netz.

Vor der ersten und letzten Note werden jeweils Nullmatrizen eingefügt. Wäre die in Abb. 43 betrachtete Note die erste Note des Kontrapunkts, würden die Scheiben A bis H nur aus Nullen bestehen.

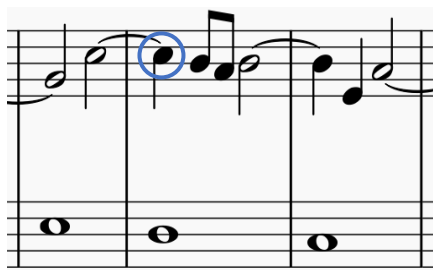


Abb. 41 Ursprüngliche Noten. Betrachtete Note umkreist.

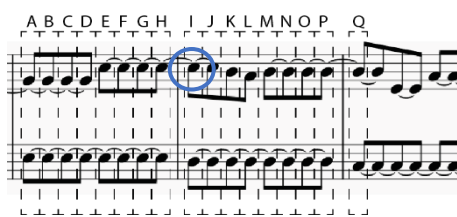


Abb. 42 Die Noten aus Abb. 41 dargestellt als Achtelnoten. Betrachtete Note umkreist.

[illegible]

Abb. 43 Die Noten aus Abb. 41 im codierten Format als Achtelnotenscheiben. Betrachtete Note mit Pfeil versehen.

6.1.1 Feedforward-Input

Für den Feedforward-Input werden die Achtelnotenscheiben komplett als Input in das Netz eingespeist. Durch die 40 Features pro Scheibe und 17 Scheiben pro Input beträgt die Zahl der Inputfeatures so $40 \cdot 17 = 680$.

$$Input_{FF} := \begin{bmatrix} A \\ \vdots \\ O \end{bmatrix}, \text{vgl. Abb. 43}$$

Netze, die mit diesem Input trainiert werden, werden im Folgenden als FF-Netze bezeichnet und verwenden das im Kapitel „Verwendete Netze“ S.28 beschriebene Feedforward-Netz.

6.1.2 Durchlaufender sequenzieller Input

Der durchlaufende sequenzielle Input ist dem Feedforward-Input sehr ähnlich. Dadurch, dass das LSTM Sequenzen als Input verwenden kann, können alle Achtelnotenscheiben nacheinander in das Netz gegeben werden. Dabei ist die Sequenzlänge $s = 17$ und jedes Sequenzelement ist eine Achtelnotenscheibe aus 40 Elementen. Bei diesem Input liegt die betrachtete Note, für die die Regel gilt, nicht am Ende, sondern in der Mitte der Sequenz.

Netze, die mit diesem Input trainiert werden, werden im Folgenden als DL-Netze bezeichnet und verwenden das im Kapitel „Verwendete Netze“ S.28 beschriebene LSTM.

$$Input_{DL} := [A \quad \cdots \quad Q], \text{ vgl. Abb. 43}$$

6.1.3 Zusammenlaufender sequenzieller Input

Der zusammenlaufende sequenzielle Input nähert sich der betrachteten Note von beiden Seiten gleichzeitig an. Um das zu erreichen, werden immer zwei Achtelnotscheiben gleichzeitig ins Netz gegeben. Dabei stellen die oberen 40 Neuronen die Achtelscheibe der linken Seite der Note dar und die unteren 40 die der rechten Seite. Insgesamt ergibt sich also ein Inputvektor mit 80 Features³².

Die Sequenz verkürzt sich so von $s = 17$ auf eine Länge von $s = 9$, wobei das letzte Sequenzelement zweimal die betrachtete Note darstellt.

$$Input_{ZL} := \begin{bmatrix} A & B & \cdots & H & I \\ Q & P & \cdots & J & I \end{bmatrix}, \text{ vgl. Abb. 43}$$

Netze, die mit diesem Input trainiert werden, werden im Folgenden als ZL-Netze bezeichnet und verwenden das im Kapitel „Verwendete Netze“ S.28 beschriebene LSTM.

6.1.4 Bewertung der Netze

Die Netze (FF-, DL- und ZL-Netz) wurden in Versuchsreihen mit den verschiedenen Daten (post-/präseparierte Mutationsdaten und stochastische Daten) trainiert. Die Beschreibungen und Ergebnisse der Versuchsreihen sind im Anhang von S. 66 (Versuchsreihe „Training mit postseparierten Mutationsdaten“) bis S. 78 (Versuchsreihe „Reduzieren der Trainingsbeispiele“) zusammengefasst. Im folgenden Kapitel werden die Ergebnisse besprochen und untereinander verglichen.

Überwiegend werden für den Vergleich der Netze der F1-Score und die *Accuracy* herangezogen. Vereinzelt werden auch die diskreten Werte der *Confusion Matrices* zur Auswertung genutzt.

In der *Confusion Matrix* ist notiert, wie oft das Netz die Regeln richtig erkennt und wie oft es einen falschen Alarm gibt. Es gibt vier unterschiedliche Fälle:

TP “true positive”: Die Regel ist gebrochen und wurde vom Netz auch als gebrochen erkannt.	FP “false positive”: Die Regel ist <u>nicht</u> gebrochen, wurde vom Netz aber als gebrochen erkannt.
FN “false negative”: Die Regel ist gebrochen, wurde vom Netz aber als <u>nicht</u> gebrochen erkannt.	TN “true negative”: Die Regel ist <u>nicht</u> gebrochen und wurde vom Netz auch als <u>nicht</u> gebrochen erkannt.

³² Für die Versuche wurde die doppelte Information, in welcher Stimme sich der *c.f.* befindet, einmal gestrichen. Dadurch ergibt sich eine Gesamtanzahl von 79 Inputfeatures.

Die *Accuracy* (Acc) bezieht die richtig erkannten Elemente (TP und TN) auf die Elemente insgesamt und ist definiert als:

$$Acc = \frac{TP + TN}{TP + FP + FN + TN}$$

Durch die teils sehr unterschiedlichen Verhältnisse zwischen gebrochenen und nicht gebrochenen Regeln eignet sich die Acc nur bedingt, um die Netze zu bewerten. Regeln wie R9 werden in den postseparierten Daten nur 127-mal gebrochen, insgesamt gibt es aber 47.702 Elemente. Würde das Netz immer ausgeben, dass die Regel nicht gebrochen ist, würde eine fast perfekte Acc von 0,997 erreicht, obwohl 100 Prozent aller regelbrechenden Noten fälschlicherweise als nicht gebrochen erkannt wurden. Sie eignet sich aber dennoch, um die Netze untereinander zu vergleichen.

Der F1-Score setzt die Genauigkeit (*precision*) und die Trefferquote (*sensitivity*³³) gegeneinander in ein Verhältnis. Die Genauigkeit ist dabei definiert durch:

$$precision = \frac{TP}{TP + FP}$$

Sie beschreibt, wie gut regelverletzende Elemente in Bezug auf die insgesamt als regelverletzend bewerteten Elemente gefunden werden. Also: „Wie oft hat das Netz behauptet, eine Regel zu erkennen und wie oft lag es dabei richtig?“.

Die Trefferquote ist definiert durch:

$$sensitivity = \frac{TP}{TP + FN}$$

Sie beschreibt, wie gut regelverletzende Elemente in Bezug auf die insgesamt regelverletzenden Elemente erkannt wurden. Also: „Wie oft ist die Regel gebrochen und wie oft hat das Netz eine gebrochene Regel erkannt?“.

Der F1-Score ist definiert durch:

$$F_1 = \frac{2}{precision^{-1} + sensitivity^{-1}} = 2 \cdot \frac{precision \cdot sensitivity}{precision + sensitivity}$$

Er bildet das harmonische Mittel aus der Genauigkeit und der Trefferquote.

Der F-Score erlaubt auch das Anpassen des Verhältnisses zwischen Genauigkeit und Trefferquote:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot sensitivity}{precision \cdot \beta^2 + sensitivity}$$

Ein Wert von $\beta > 1$ würde der Trefferquote eine höhere Relevanz zusprechen. Der F_β -Score wird hier allerdings nur der Vollständigkeit halber erwähnt; in dieser Arbeit wird ausschließlich $\beta = 1$, also der F1-Score verwendet.

³³ Die *sensitivity* wird in der Literatur auch als *recall* bezeichnet.

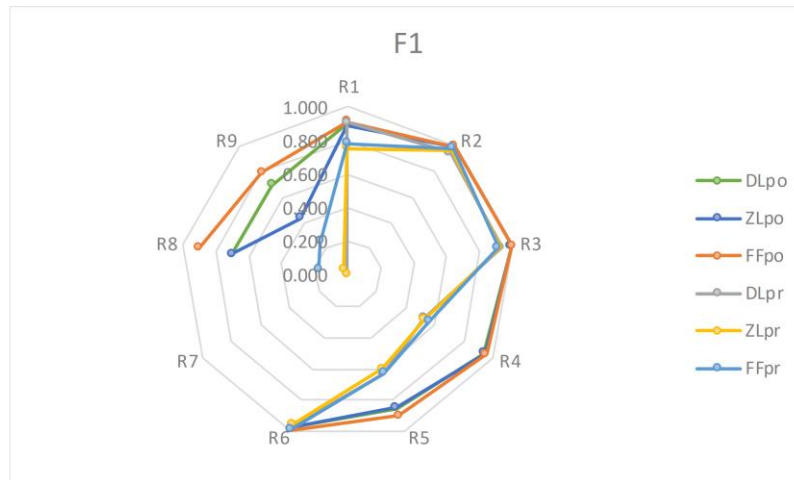
6.2 Training mit prä- und postseparierten Mutationsdaten

Werden für das Training der Netze die postseparierten Mutationsdaten verwendet, können – getestet auf den postseparierten Validationsdaten – gute bis sehr gute Ergebnisse erzielt werden (vgl. Abb. 44, F1 postsepariert). Wie in Kapitel „Gleiche Datensätze“ S. 24 gezeigt wurde, enthalten diese Daten aber viele gleiche Elemente. Es ist also anzunehmen, dass die Netze nur bereits gesehene Elemente wiedererkennen (vgl. A2 Versuchsreihe „Training mit postseparierten Mutationsdaten“ S. 66).

Werden die postseparierten Daten für das Training verwendet, können keine so guten Ergebnisse mehr erreicht werden. Die Regeln R8 und R9 können von den LSTM-Netzen nicht mehr bewertet werden und auch das FF-Netz liefert keine zufriedenstellenden Ergebnisse (vgl. Abb. 44, F1 präsepariert). Auch bei allen anderen Regeln schneiden die mit den postseparierten Daten trainierten Netze besser ab (vgl. Abb. 44, post/prä max.).

Für die Bedeutung der Abkürzungen wie „DLpo“ wird auf die Nomenklatur S. 58 verwiesen.

Die Ergebnisse der jeweiligen Versuchsreihen werden in Anhang A S. 66 bis 81 beschrieben. An dieser Stelle werden die Ergebnisse untereinander verglichen.



	F1 postsepariert			F1 präsepariert			post	prä
	DLpo	ZLpo	FFpo	DLpr	ZLpr	FFpr	max.	max.
R1	0.900	0.890	0.911	0.907	0.754	0.782	0.911	0.907
R2	0.996	0.998	0.998	0.953	0.971	0.983	0.998	0.983
R3	0.995	0.996	0.997	0.935	0.922	0.911	0.997	0.935
R4	0.942	0.946	0.954	0.527	0.535	0.562	0.954	0.562
R5	0.853	0.847	0.899	0.626	0.603	0.628	0.899	0.628
R6	0.966	0.968	0.990	0.967	0.952	0.986	0.990	0.986
R7*	-	-	-	-	-	-	-	-
R8	0.698	0.699	0.900	0.000	0.000	0.172	0.900	0.172
R9	0.699	0.435	0.791	0.000	0.032	0.253	0.791	0.253
Σ	7.049	6.779	7.441	4.915	4.769	5.277	7.442	5.426

Abb. 44 Der F1-Score der mit den post- und präseparierten Daten trainierten neuronalen Netze im direkten Vergleich. Die Farbcodierung bezieht sich immer auf eine Zeile (8 Werte). Rechts die besten Ergebnisse der jeweiligen Versuchsreihen. Das Netzdiagramm zeigt die F1-Scores, das perfekte Netz würde alle 9 Ecken im Punkt 1,000 schneiden.

*Für R7 kann keine Aussage getroffen werden, da diese Regel in den post- und präseparierten Validationsdaten nicht verletzt wird.

Für R1, R2, R3 und R6 können passable Ergebnisse erzielt werden. Dass R1 erkannt wird, kann so gewertet werden, dass das Netz erkennt, ob es sich um die erste bzw. letzte Note im Kontrapunkt handelt. Allerdings kann R1 „Start- und Endregel“:

$$P(I_t | N_{t-1} = \emptyset \vee N_{t+1} = \emptyset)$$

schon erkannt werden, wenn nur die Achtelnotenscheibe vor, bzw. nach, der betrachteten Note komplett aus Nullen besteht.

Interessant ist das gute Ergebnis beim Lernen von R3 „Bevorzugte *Intervals*“:

$$P(I_t | Tied(N_t) \wedge Pos(N_t) \wedge I_{t+1} \wedge S_{t+1} \wedge N_{t-1} \wedge N_{t+1})$$

Diese Regel hängt unter anderem vom Intervall der Nachfolgenote ab. Außerdem muss das Sprungintervall zur Nachfolgenote erkannt werden. Für eine ganze Note müsste also ein Bereich von 9 Achtelnotenscheiben betrachtet werden. Regel R3 wird in den Validationsdaten 7.399 Mal gebrochen und enthält 2.488 Elemente, die im Bereich von 9 Achtelnotenscheiben (davon nur 4 Achtelnotenscheiben, die die Nachfolgenote abbilden würden) auch so in den Trainingsdaten vorkommen.³⁴

Das ZL-Netz kann für R3 7.357 Elemente als TP erkennen. Es ist also anzunehmen, dass tatsächlich das Konzept einer Nachfolgenote gelernt wurde.

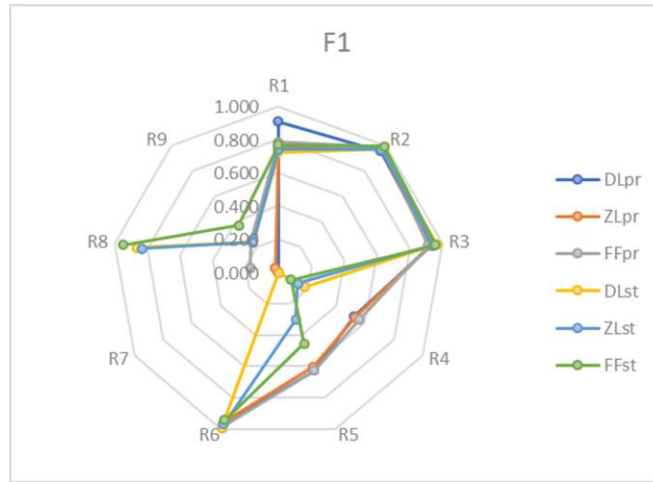
Trainingsdaten, die Ähnlichkeit mit den prä- und postseparierten Daten haben, wären in der Praxis am einfachsten zu erstellen. Solche Daten könnten beispielsweise aus bewerteten Klausuraufgaben stammen und müssten nur in ein Format gebracht werden, mit dem die Netze trainiert werden können.

Da die hier verwendeten Beispiele allerdings nur auf 6 verschiedenen *cantus firmi*, und 51 Kontrapunkten beruhen, müssten erst weitere Versuche unternommen werden, um den Lernerfolg (für Regeln R1, R2, R3 und R6) durch solche Daten zu bestätigen.

³⁴ Im 5-Achtelnotenbereich 3.769 Elemente.

6.3 Training mit stochastischen Trainingsdaten

Wird mit den stochastischen Daten trainiert und auf den postseparierten Daten getestet, können für einzelne Regeln gute F1-Scores erzielt werden (R2, R3, R6 und R8). Auch für Regel R9 können bessere Ergebnisse erzielt werden, allerdings liefern die Netze keine Ergebnisse, die sich in der Praxis verwenden lassen würden. Für R9 stehen für das mit den stochastischen Daten trainierte DL-Netz 128 TP's gegen 822 FP's (für das FF: 119 TP gegen 397 FP). Es werden also öfter Daten fälschlicherweise als regelverletzend bewertet als richtigerweise.



	F1 präsepariert (pr)			F1 stochastisch (st)			prä	stoch
	DLpr	ZLpr	FFpr	DLst	ZLst	FFst	max.	max.
R1	0.907	0.754	0.782	0.723	0.740	0.768	0.907	0.768
R2	0.953	0.971	0.983	0.964	0.964	0.988	0.983	0.988
R3	0.935	0.922	0.911	0.968	0.941	0.957	0.935	0.968
R4	0.527	0.535	0.562	0.178	0.133	0.087	0.562	0.178
R5	0.626	0.603	0.628	0.002	0.304	0.454	0.628	0.454
R6	0.967	0.952	0.986	0.994	0.970	0.945	0.986	0.994
R7*								
R8	0.000	0.000	0.172	0.863	0.831	0.950	0.172	0.950
R9	0.000	0.032	0.253	0.237	0.237	0.370	0.253	0.370
Σ	4.915	4.769	5.277	4.929	5.122	5.519	5.277	5.519

Abb. 45 Der F1-Score der mit den prä- und stochastischen Daten trainierten neuronalen Netze im direkten Vergleich. Die Farbcodierung bezieht sich immer auf eine Zeile (8 Werte). Rechts die besten Ergebnisse der jeweiligen Versuchsreihen. Das Netzdiagramm zeigt die F1-Scores.

*Für R7 kann keine Aussage getroffen werden, da diese Regel in den prä- und postseparierten Validationsdaten nicht verletzt wird.

Regel R4 „Einleiten der Schlussnote“:

$$P(I_t | N_{t+2} = \emptyset \wedge \text{bass}(c.f.)),$$

$$P(I_t | N_{t+3} = \emptyset \wedge \text{bass}(c.f.) = \text{true})$$

kann mit den stochastischen Daten nicht mehr gut gelernt werden. In den postseparierten Validationsdaten kommen 32 Elemente vor, die auch in den stochastischen Trainingsdaten vorhanden sind (bezogen auf den 5-Achtnotenbereich). Insgesamt wird die Regel 689-mal gebrochen und vom DL-Netz 109-mal als TP erkannt – demgegenüber stehen allerdings 580 FN und 430 FP.

R4 besteht aus zwei Teilregeln, von denen eine nur angewandt wird, wenn der *c.f.* in der unteren Stimme steht. Wie zu Beginn dieses Kapitels beschrieben, sehen die Netze den *c.f.* immer an derselben Stelle, und es wird ihnen nur anhand eines Inputfeatures mitgeteilt, ob der *c.f.* in der unteren oder oberen Stimme steht. Die Netze müssten also eine Art „Kippschalterfunktion“ erlernen, um im Anschluss zu bestimmen, ob die Regeln gebrochen sind oder nicht. Hinzu kommt, dass die Netze nicht genau bestimmen können, ob es sich um die vorletzte bzw. vorvorletzte Note handelt – sie sehen unter Umständen das Ende des Kontrapunkts noch nicht.³⁵

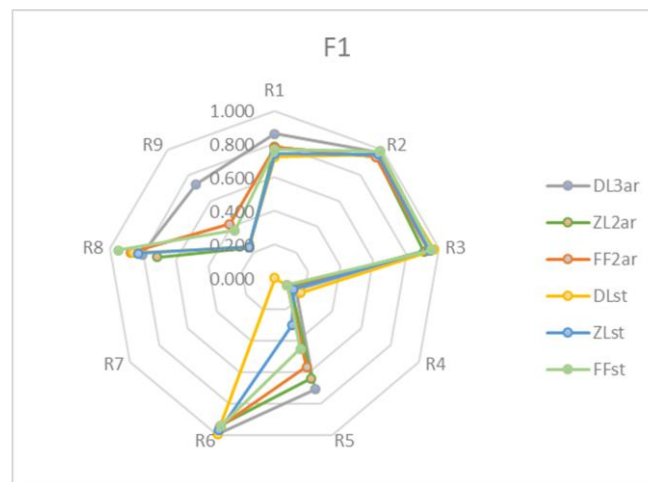
Die Ergebnisse zeigen, dass auch mit Daten gelernt werden kann, die sehr verschieden von handgeschriebenen Daten sind. Im Folgenden werden nur noch die stochastischen Daten (oder eine Selektion aus ihnen) als Trainingsdaten genutzt, da hier die Chance am geringsten ist, dass die Lernerfolge auf mehrfach vorhandene Datensätze zurückzuführen wären.

³⁵ Ein einzelner Versuch, der zu keiner der Versuchsreihen gehört, liefert Ergebnisse, die diese Annahme verstärken. Ein DL-Netz, das mit einer Sequenz, die einen Bereich von 49 Achtelnotenscheiben abdeckt, trainiert wurde, konnte einen F1-Score von 0,521 erreichen. (Einzelversuch S. 45)

6.4 Bewertung mehrerer Regeln

Da die musiktheoretischen Voraussetzungen – also welches Intervall eine Note mit einer anderen bildet oder mit welcher Bewegung zu einer Note geschritten wird – für viele Regeln ähnlich sind, könnte das Erlernen einer Regel auch positive Folgen für das Erlernen einer zweiten Regel haben.

Um diese Theorie zu überprüfen, wurden die Netze darauf trainiert, alle Regeln gleichzeitig zu bewerten.



	F1 alle Regeln (ar)			F1 stochastisch (st)			ar	stoch
	DL3ar	ZL2ar	FF2ar	DLst	ZLst	FFst	max.	max.
R1	0.861	0.784	0.780	0.723	0.740	0.768	0.861	0.768
R2	0.978	0.949	0.942	0.964	0.964	0.988	0.978	0.988
R3	0.949	0.910	0.938	0.968	0.941	0.957	0.949	0.968
R4	0.147	0.113	0.089	0.178	0.133	0.087	0.147	0.178
R5	0.708	0.641	0.565	0.002	0.304	0.454	0.708	0.454
R6	0.989	0.941	0.948	0.994	0.970	0.945	0.989	0.994
R7*								
R8	0.806	0.713	0.871	0.863	0.831	0.950	0.871	0.950
R9	0.730	0.237	0.420	0.237	0.237	0.370	0.730	0.370
Σ	6.169	5.287	5.553	4.929	5.122	5.519	6.169	5.519

Abb. 46 Der F1-Score der mit den stochastischen Daten trainierten neuronalen Netze im direkten Vergleich. Abgebildet sind die in der Summe ihrer F1-Scores am besten abschneidenden Netze. Die Farbcodierung bezieht sich immer auf eine Zeile (8 Werte). Rechts die besten Ergebnisse der jeweiligen Versuchsreihen. Das Netzdiagramm zeigt die F1-Scores.
*Für R7 kann keine Aussage getroffen werden, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.

Obwohl das gleichzeitige Bewerten mehrerer Regeln teils einen positiven Effekt auf den F1-Score hat und im Fall von Regel R9 und R5 erst das Lernen aus den wenigen Beispieldaten ermöglichte, hat diese Methode auch Nachteile (vgl. Abb. 46, ar max., stoch max.).

Die alle Regeln lernenden Netze erreichen für R8 nicht so hohe F1-Scores wie ihre eine Regel bewertenden Pendanten.

Die erhoffte Verbesserung für das Lernen von R5 tritt allerdings tatsächlich ein. Das DL3ar-Netz kann in den postseparierten Validationsdaten 822 TPs erkennen. Dagegen stehen 411 FPs und 266 FNs. Dies ist kein perfektes Ergebnis, aber die Methode, über andere Regeln die Muster zu erkennen und dann auf eine andere Regel zu übertragen, scheint zu greifen.

Würde ein Programm entwickelt, das mit den hier beschriebenen Techniken lernt, Regeln zu bewerten, kann keine klare Empfehlung gegeben werden, mit welchem Ansatz trainiert werden soll. Es sollte im Einzelfall geprüft werden, ob eine Regel besser allein oder zusammen mit anderen Regeln erlernt werden kann.

Das sollte auch bei der Architektur eines entsprechenden Programms berücksichtigt werden. Ein Netz, das auf alle Regeln trainiert wird, sollte auch dazu eingesetzt werden können, einzelne Regeln zu bewerten.

6.5 Reduzieren der Trainingsbeispiele

Sollen die beschriebenen Netze in der Lehre eingesetzt werden und neue Regeln lernen, müssen Trainingsdaten für diese Regeln produziert werden. Die bisherigen Netze wurden mit von einem Algorithmus generierten und bewerteten Daten trainiert. Existiert bereits eine Methode, die leicht zu implementieren ist und mit einhundertprozentiger Wahrscheinlichkeit bestimmen kann, ob eine Regel gebrochen oder erfüllt ist, liegt es nahe, diese Methode zu nutzen.

Die neuronalen Netze können nicht garantieren, dass jedes Beispiel richtig bewertet wird. Der Vorteil liegt darin, dass sie ohne Vorkenntnisse trainiert werden können, wenn ein nutzerfreundliches Programm zur Verfügung gestellt wird.

Nicht jeder Professor einer musikalischen Fakultät hat die Lust und Zeit, sich mit der Programmierung von Algorithmen zu beschäftigen. Ein Programm, in dem nur korrigierte Beispieldaten bereitgestellt werden müssen, kann mit weniger Zeitaufwand verwendet werden.

Hinzu kommt, dass die Beispieldaten, die bereitgestellt werden müssten, unter Umständen durch bewertete Klausuraufgaben schon existieren. Da diese Daten allerdings in ein Format gebracht werden müssen, das die neuronalen Netze verstehen und nicht unendlich viele bewertete Daten vorliegen, soll in diesem Abschnitt geprüft werden, wie weit die Anzahl der benötigten Negativbeispiele reduziert werden kann.

Um sowohl ein Netz testen zu können, das alle Regeln lernt, als auch solche, die mehrere Regeln lernen, wird eine Trainingsdatenmenge erstellt, die dann für alle Netze und für alle Regeln gilt. Als ein Anhaltspunkt wurden 1.000 Kontrapunkte gewählt.

Für die Versuche wurden aus den stochastischen Trainingsdaten 1.000 unterschiedliche Kontrapunkte ausgewählt, in denen mindestens eine Note gegen Regel R8 verstößt. R8 ist die Regel, gegen die in den Daten am seltensten (1.141-mal) verstoßen wird. (Abb. 47)

Anschließend wurden aus den übrigen Trainingsdaten Kontrapunkte, in denen gegen R5 verstoßen wurde, ausgewählt, bis gegen diese Regel auch mindestens 1.000-mal verstoßen wurde. Die so erzeugten Trainingsdaten bestehen aus 1.233 Kontrapunkten. (Abb. 48)

Regel	Gebrochen
R1	1592
R2	9422
R3	7676
R4	945
R5	733
R6	8349
R7	5053
R8	1091
R9	3679

Abb. 47 Nach der ersten Auswahl sind in den 1.000 Ausgewählten sheets alle Regeln über 1.000-mal gebrochen außer R4 und R5. Gegen R8 wird 1.091-mal verstoßen, da in einigen Kontrapunkten mehrere Noten gegen sie verstoßen.

Regel	Gebrochen
R1	1926
R2	11592
R3	9463
R4	1167
R5	1055
R6	10250
R7	6082
R8	1091
R9	4548

Abb. 48 Nach der 2. Auswahl sind in insgesamt 1.233 sheets alle Regeln über 1.000-mal gebrochen.

Die Verteilung der c.f.-Notenhöhen ist in diesen Daten nicht gleichverteilt (Abb. 49). Sie ergibt sich aus den Regeln, nach denen ausgewählt wurde. Eine Teilregel von R8 ist „Tied notes should not be resolved from a seventh to an octave, if c.f. is in soprano“³⁶, sie wird im Verhältnis 5:2 öfter gebrochen als die Teilregeln, bei denen der c.f. in der unteren Stimme steht, woraus sich der höhere Anteil in den Noten A3 bis C6 ergibt.

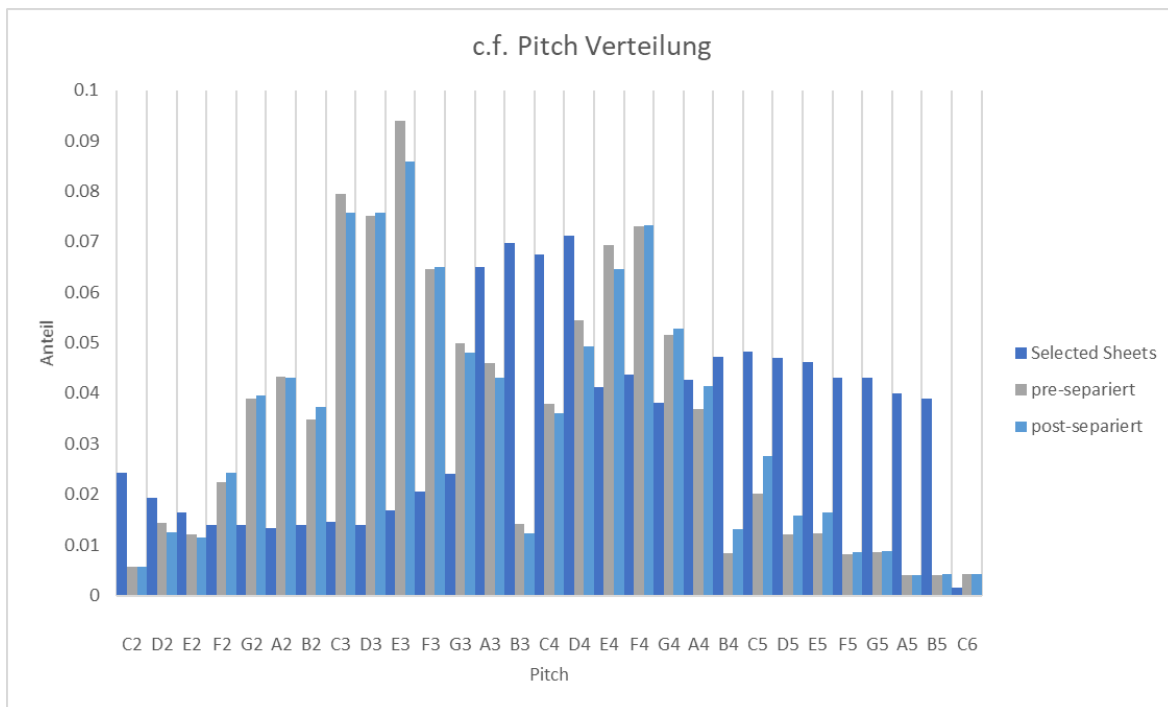


Abb. 49 Die Verteilung der c.f.-Noten in den ausgewählten Daten ähnelt einer Sinuskurve mit einem Sprung in der Mitte, ergibt sich aber aus den Wahrscheinlichkeiten, mit denen der c.f.-Generator arbeitet.

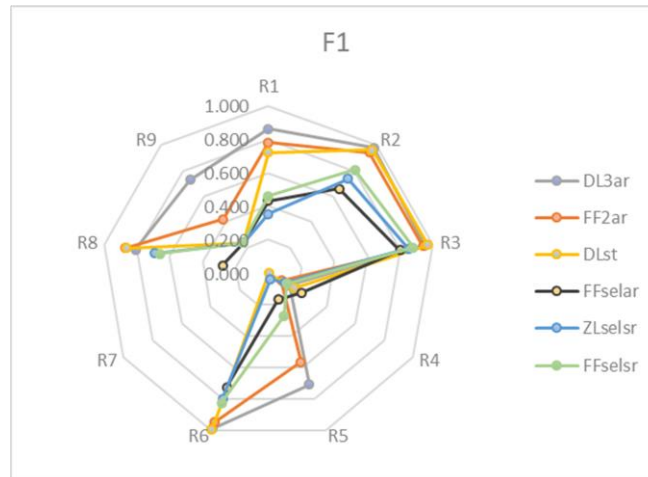
Der Anstieg in der Mitte ergibt sich dadurch, dass die Noten A3 bis D4 sowohl für die untere als auch für die obere Stimme generiert werden können.

Die Verteilung in der unteren Stimme (C2 bis D4) ergibt sich aus der Teilregel von R8 „Tied notes should not be resolved from an unison to a second, if c.f. is in bass“. Das Intervall einer perfekten 1 (unison) kann nur in den Noten stattfinden, die in beiden Stimmen generiert werden können (A3 bis D4). Eine Eigenschaft des Generators ist, von einer Note in $\frac{4}{5}$ der Fälle nur den Sprung eines Zweier- oder Dreierintervalls zu generieren. Eine weitere Eigenschaft ist: sollte die höchstmögliche Note in der Stimme (D4) überschritten werden, wird die Note am unteren Ende der Stimme (C2) eingefügt. D4 hat also eine hohe Wahrscheinlichkeit, generiert zu werden und gleichzeitig R8 zu brechen, da zu ihr in einem Zweier- oder Dreierintervall von A3, B3 und C3 gesprungen werden kann. Genauso verhält es sich mit C2, da dorthin von einem B3, C3, und D4 gesprungen werden kann. Noten wie C3, die nicht am Ende der Stimme stehen, können nur in $\frac{1}{5}$ der Fälle generiert werden und R8 verletzen.

³⁶ Diese Regel kann nur gebrochen werden, wenn der c.f. in der oberen Stimme (Soprano) ist. Im Anhang, unter „R8 Verbotene Suspensions“ S. 63, steht die Funktion, mit der die Regel geprüft wird.

Ergebnisse

Mit den selektierten Daten können – wie zu erwarten – nicht mehr so gute F1-Scores erreicht werden wie mit den gesamten stochastischen Trainingsdaten (vgl. Abb. 50, Netzdiagramm und Tab. stoch max. sel. max.). Die Ergebnisse für R2, R3 und R6 befinden sich noch in einem annehmbaren Bereich.



	F1 stochastisch			F1 selektiert			stoch	sel
	DL3ar	FF2ar	DLst	FFsalar	ZLselsr	FFselsr	max.	max.
R1	0.861	0.780	0.723	0.433	0.355	0.460	0.861	0.460
R2	0.978	0.942	0.964	0.659	0.737	0.805	0.978	0.805
R3	0.949	0.938	0.968	0.798	0.850	0.876	0.968	0.876
R4	0.147	0.089	0.178	0.230	0.110	0.125	0.178	0.230
R5	0.708	0.565	0.002	0.169	0.037	0.271	0.708	0.271
R6	0.989	0.948	0.994	0.725	0.800	0.824	0.994	0.824
R7*								
R8	0.806	0.871	0.863	0.278	0.694	0.662	0.871	0.694
R9	0.730	0.420	0.237	0.237	0.236	0.239	0.730	0.239
Σ	6.169	5.553	4.929	3.529	3.819	4.262	6.169	4.262

Abb. 50 Der F1-Score der mit den stochastischen und selektierten Daten trainierten neuronalen Netze im direkten Vergleich. Abgebildet sind die in ihrer Summe der F1-Scores am besten abscheidenden Netze. Die Farbcodierung bezieht sich immer auf eine Zeile (8 Werte). Rechts die besten Ergebnisse der jeweiligen Versuchsreihen. Das Netzdiagramm zeigt die F1-Scores.
*Für R7 kann keine Aussage getroffen werden, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.

Zu bemerken ist, dass für R4 der beste F1-Score aller Versuchsreihen erreicht werden konnte.³⁷ Wie in 6.3 „Training mit stochastischen Trainingsdaten“ beschrieben, müsste um R4 zu erlernen eine Art „Kippschalter“ erlernt werden, der beurteilt, ob der *c.f.* in der unteren oder oberen Stimme steht. Die Verbesserung könnte durch die Auswahl der selektierten Daten entstehen. Sie bestehen zu einem Großteil aus Kontrapunkten, die nach R8 ausgewählt wurden (für die ebenfalls entschieden werden

³⁷ Mit Ausnahme der Ergebnisse des Einzelversuchs (S. 45), bei dem mit einem 49 Achtelnoten umfassenden Bereich gearbeitet wird.

muss, in welcher Stimme der *c.f.* steht), beinhalten also im Verhältnis mehr Beispiele, die mit der Stimmposition zu tun haben.

Durch diese Versuchsreihe konnte festgestellt werden, dass das Lernen aus wenigen Daten möglich ist, aber mit einer reduzierten Genauigkeit und Trefferquote zu rechnen ist.

Zu beachten ist allerdings, dass die hier verwendeten Kontrapunkte sehr viele Negativbeispiele bereitstellen. Ein einfaches Verwenden von Daten, die zum Beispiel über Klausuren erhoben wurden, würde also wahrscheinlich nicht zu einem vergleichbaren Trainingserfolg der neuronalen Netze führen – es sei denn, die Musikstudenten hätten allesamt nicht für die Klausuren gelernt.

6.6 Diskussion der Ergebnisse

Mit den verschiedenen Trainingsdaten können die neuronalen Netze unterschiedlich gute Ergebnisse erzielen. Zwischen LSTM-Netzen und Feedforward-Netzen gibt es beim Lernen einzelner Regeln keinen Favoriten, der diese Disziplin in allen Bereichen deutlich dominiert (vgl. Abb. 51, F1 A3).

Die Regel R8 kann jeweils in A3 Versuchsreihe „Training mit präseparierten Mutationsdaten“ besser von einem Feedforward-Netz erkannt werden und in A5 Versuchsreihe „Training einzelner Netze auf alle Regeln“ besser von einem LSTM basierten Netz (vgl. Abb. 51,).



Abb. 51 Der F1-Score einzelner Netze der Versuchsreihen A3 und A5 im direkten Vergleich. Abgebildet sind die in der Summe ihrer F1-Scores am besten abschneidenden Netze. Die Farbcodierung bezieht sich immer auf eine Zeile und Versuchsreihe (2 Werte). Die Netzdiagramme zeigen die F1-Scores. A3 und A5 beziehen sich jeweils auf die Versuchsreihen A3 Versuchsreihe „Training mit präseparierten Mutationsdaten“ und A5 Versuchsreihe „Training einzelner Netze auf alle Regeln“.

*Für R7 kann keine Aussage getroffen werden, da diese Regel in den jeweiligen Validationsdaten nicht verletzt wird.

Beim Lernen aller Regeln in A5 Versuchsreihe „Training einzelner Netze auf alle Regeln“ scheint die LSTM-Architektur allerdings von Vorteil zu sein. Die Regeln R1 und R5 sind von diesem Vorteil besonders betroffen.

Regel R1 Start- und Endregel:

$$P(I_t | N_{t-1} = \emptyset \vee N_{t+1} = \emptyset)$$

und R5 Bevorzugte *Motion*-abhängige *Intervals*:

$$P(I_t | M_t \wedge I_{prevDBeat})$$

beurteilen beide das Intervall der betrachteten Note. Für R1 muss zusätzlich erkannt werden, ob vor oder hinter der Note keine weitere Note ist, für R5 muss das Intervall der vorherigen *Downbeat* Note erkannt werden. Beides sind Eigenschaften, die sich auf die Startposition der vorherigen *c.f.*-Note („new_note“ vgl. Abb. 30, S. 23) beziehen können.

Vielleicht ist die Notenposition ein Konzept, das von LSTM-basierten Netzen einfacher erlernt werden kann.

Ein Einzelversuch, bei dem getestet wurde, ob Regel R4 besser erlernbar ist, wenn der Achtelnotenscheibenbereich von 17 auf 49 Scheiben erhöht wird, ergab, dass zwar speziell für Regel R4 bessere Ergebnisse vom LSTM-basierten Netz erzielt werden können, beim Lernen aller Regeln aber das Feedforward-Netz besser abschneidet.

Das widerspricht der auf der vorherigen Seite gewonnenen Erkenntnis, dass die LSTM-Netze besser zum Lernen aller Regeln geeignet sind. Es müssten weitere Versuche unternommen werden, um hier eine klare Aussage treffen zu können.

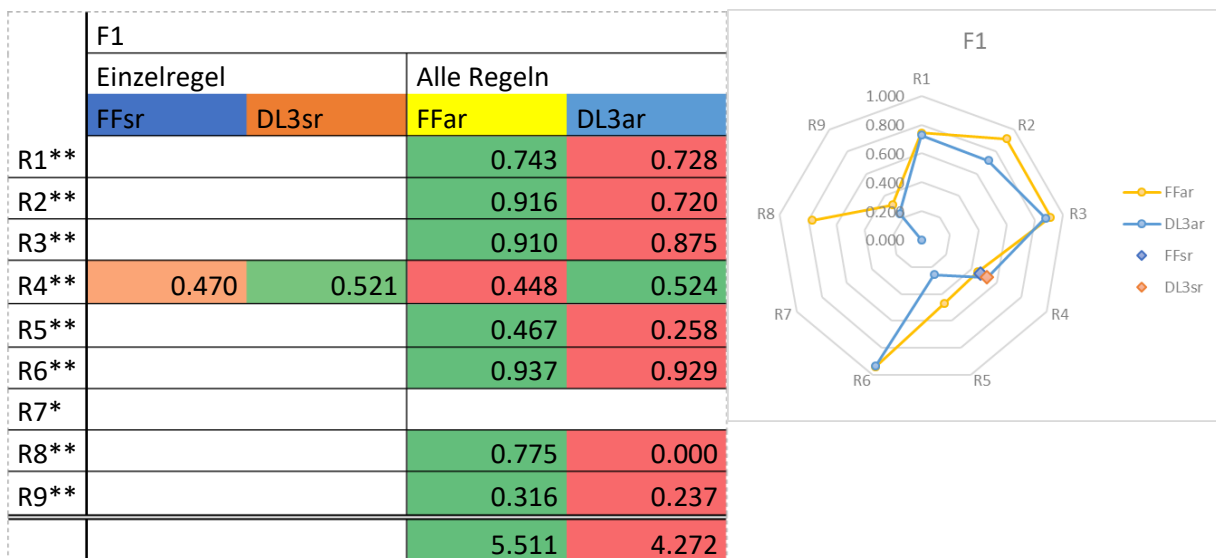


Abb. 52 Der F1-Score eines Einzelversuchs, bei dem Netze auf Regel R4 trainiert wurden, der Achtelnotenscheibenbereich aber von 17 auf 49 erhöht wurde. Die Farbcodierung bezieht sich immer auf eine Zeile (2 bzw. 4 Werte). Das Netzdiagramm zeigt die F1-Scores.

Trainingsdaten: stochastische Trainingsdaten, Validationsdaten: postseparierte Validationsdaten. Netzeinstellungen: FF wie in A2, DL3sr/ar wie in A5.

*Für R7 kann keine Aussage getroffen werden, da diese Regel in den jeweiligen Validationsdaten nicht verletzt wird.

**Für die Einzelregeln wurden nur Netze trainiert, die R4 bewerten können.

Die LSTM-basierten Netze benötigen im Input-Layer zwar weniger Gewichte, der Overhead ist aber so groß, dass die trainierbaren Parameter insgesamt mehr sind als bei den Feedforward-Netzen. In dem Versuch mit 49 Achtelnotenscheiben wurden zwei Netze mit den Einstellungen aus Abb. 54 trainiert.

Mit 1.960 Inputneuronen besitzt das FFsr insgesamt 709.801 trainierbare Parameter. Das LSTM besitzt insgesamt 4.777.969 trainierbare Parameter.

Die Anzahl der Parameter bleibt beim LSTM zwar gleich, aber erst ab einem betrachteten Bereich von 392 Achtelnotenscheiben³⁸ würde das Feedforward-Netz mehr trainierbare Parameter besitzen: Mit 15.680 Input-Neuronen 4.990.441 trainierbare Parameter.³⁹

Das Training des Feedforward-Netzes benötigte 31:35 Minuten, das des LSTM-Netzes 2:49:44 Stunden. Die Trainingsdauer ist also ein Faktor, der beachtet werden sollte, wenn das Programm an Musikprofessoren ausgehändigt werden soll, die vielleicht nicht bereit sind, lange auf Ergebnisse zu warten.

	FFsr	DL3sr
i	1960	40
h	312	624
o	1	1
hl	2	2
b	50	50
ep	30	30
do	0.0	0.3
lr	0.0001	0.0001

Abb. 53 Einstellungen der Netze. Für die ar-Netze ändert sich nur die Anzahl der Output-Neuronen auf 9.

i: Input-Neuronen, h: hidden Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainierte.

³⁸ Das entspricht im $\frac{3}{4}$ -Takt 65,3 Takten. Johann Sebastian Bachs Menuett in G-Dur besitzt 32 Takte, ließe sich also 2-mal in einem so großen Input abbilden.

³⁹ Die LSTM-Netze mit 2 Hidden Layern und 312 Hidden-Layer-Neuronen besitzen 1.220.857 trainierbare Parameter. Ein Feedforward-Netz würde hier ab einem Input von 91 Achtelnotenscheiben mit 3.640 Input-Neuronen mehr trainierbare Parameter (1.233.961) besitzen. Im $\frac{3}{4}$ -Takt wären das 15,16 Takte, also nur die ungefähre Hälfte des Menuetts.

Fazit

In dieser Arbeit wurde untersucht, ob die Regeln des Gradus ad Parnassum von neuronalen Netzen erlernt werden können. Es wurden fünf Versuchsreihen und eine Analyse der Trainings-, Test- und Validationsdaten durchgeführt, um ein besseres Verständnis davon zu erhalten, wie die neuronalen Netze arbeiten.

Durch die in dieser Arbeit verwendete diskrete Unterscheidung der Regeln ist es möglich, dem Musikstudenten⁴⁰ ein direktes Feedback⁴¹ zu hinterlassen. So kann die KI dem Studenten nicht nur mitteilen, dass eine Regel gebrochen wurde, sondern auch angezeigt werden, was diese Regel bedeutet.

Für die Analyse der Datensätze auf gleiche Elemente konnte eine effiziente Lösung gefunden werden, auch große Mengen an Daten zu vergleichen.⁴²

Durch die Versuche konnte gezeigt werden, dass neuronale Netze teils schon mit wenigen Trainingsdaten in der Lage sind, zufriedenstellende Ergebnisse zu erzielen.⁴³

Die Versuche, bei denen jeweils ein Netz pro Regel angelernt wurde, konnten dabei zeigen, dass es möglich ist, ein Netz auf einzelne spezifische Muster der Musiktheorie zu trainieren.⁴⁴ Dabei wurde der Ansatz eines modularen Konzepts für ein Lernprogramm verfolgt, bei dem neue Inhalte nach und nach hinzugefügt werden können.

Die Versuche zum Erlernen aller Regeln durch jeweils ein Netz konnten zeigen, dass Regeln existieren, die besser zusammen mit anderen Regeln gelernt werden können.⁴⁵ Für ein Lernprogramm, dem einzelne neue Komponenten⁴⁶ hinzugefügt werden sollen, kann es also auch sinnvoll sein zu testen, ob die neue Komponente den Trainingserfolg der alten Komponenten noch verstärkt.

Es konnte gezeigt werden, dass keine generelle Aussage darüber getroffen werden kann, ob ein LSTM- oder ein Feedforward-Netz besser geeignet ist, die Kontrapunktregeln zu erlernen. Beide Architekturen konnten für einzelne Regeln bessere Ergebnisse erzielen.⁴⁷

Es lässt sich festhalten, dass neuronale Netze in der Lage sind, die Kontrapunktregeln nach Fux zu erlernen.

⁴⁰ Als User eines Programms, das die in dieser Arbeit beschriebenen Techniken nutzt.

⁴¹ Einer Regel kann eine Nachricht angeheftet werden wie „R9 Achtelnoten dürfen nur auf Weak Beats verwendet werden“.

⁴² 4.5.1 Gleiche Datensätze.

⁴³ A6 Versuchsreihe „Reduzieren der Trainingsbeispiele“. Regeln R2, R3 und R8 konnten in den Versuchen mit reduzierten Trainingsdaten einen F1-Score von über 0,80 erreichen.

⁴⁴ A4 Versuchsreihe „Training mit stochastischen Daten“

⁴⁵ A5 Versuchsreihe „Training einzelner Netze auf alle Regeln“

⁴⁶ Im Sinne von „erlernbare Regel / Regeln“.

⁴⁷ 6.6 Diskussion der Ergebnisse S. 44.

Ausblick

Die vorangegangene Arbeit [Scholzen, 2019] konnte zeigen, dass es möglich ist, die Kontrapunktregeln als mathematische Funktionen zu formulieren und diese durch Algorithmen auszudrücken. Durch die Eigenschaft der Netze, allein aus Beispielen zu lernen, ist es selbst jemandem, der keinen Hintergrund in der Informatik hat, möglich, eigene Regeln zu formulieren und in ein Programm zu integrieren – es müssen nur genug Trainingsdaten gesammelt werden.

Diese Daten zu beschaffen, ist vermutlich die größte Herausforderung, sollte die in dieser Arbeit beschriebene Technik weiter erforscht werden.

Im Jahr 2016 schlossen allein in Deutschland 5.153 Studenten ein Studium mit musikalischem Bezug im 1. Studienfach ab [MIZ, Deutsches Musikinformationszentrum, 2019]. Jeder dieser Studenten hat im Verlauf seines Studiums potenzielle Lerndaten geschaffen. Ein Projekt, das die Fakultäten in Kontakt bringt und das Sammeln dieser Daten veranlasst, wäre von großem Wert für die KI-Forschung in der Musikanalyse.

Lehrt eine Künstliche Intelligenz, die nicht mit einhundertprozentiger Sicherheit sagen kann, ob eine bestimmte Regel eingehalten wird oder nicht, ist es wichtig, dass der Musikstudent sich keine Fehlinformationen einprägt. Die Musik bietet hier durch ihr sehr direktes Feedback – etwas hört sich gut oder schlecht an – ein gutes Grundgerüst, die Skepsis eines Musikstudenten zu wecken und die Entscheidungen der Künstlichen Intelligenz zu hinterfragen.

Bei Programmen, die mit Künstlichen Intelligenzen arbeiten, sollte besonders darauf geachtet werden, dass der Musikstudent weiß, dass die KI auch Fehler machen kann. Zusätzlich wäre es vorteilhaft, wenn der Musikstudent solche Fehler ohne viel Aufwand melden könnte. So kann das nicht erkannte Negativbeispiel direkt gesammelt und der KI in ihren Trainingsdatensatz eingespeist werden.

Um dem Laien tatsächlich zu ermöglichen, eine KI auf neue Regeln zu trainieren, müsste ein System geschaffen werden, das es dem User ermöglicht, diese neuen Regeln zu formulieren und anschließend in Notenblättern anzuzeichnen, welche Noten welche Regeln verletzen.

Die in dieser Arbeit besprochenen Netze wurden bis jetzt nur auf die Regeln zum zweistimmigen Kontrapunkt untersucht. Wird der vorgestellte Input leicht modifiziert, könnten die Netze auch auf mehrstimmige Kontrapunkte und andere Musikstücke trainiert werden. Hierfür müssten natürlich erst die nötigen Trainingsdaten gesammelt werden. Besonders die LSTM-Netze könnten darauf untersucht werden, ob sie sich dazu eignen, auf Regeln zu testen, die nur geprüft werden können, wenn ein längerer Ausschnitt eines Musikstücks analysiert wird. Ein Beispiel wäre zu prüfen, ob ein Stück einen Klimax enthält.

Die KI-Forschung auf dem Gebiet der Musikanalyse bietet noch ein breites Forschungsfeld, das es zu erschließen gilt.

Literaturverzeichnis

- Adiloglu, K., & Alpaslan, F. N. (April 2007). A machine learning approach to two-voice counterpoint composition. *Knowledge-Based Systems Volume 20, Issue 3*, S. 300-309.
- Dada, A. (2018). *Long Short-Term Memory zur Generierung von Musiksequenzen*. Von https://www.ini.rub.de/upload/file/1521461530_7126db755dc03bec85b1/dada-bsc.pdf abgerufen
- Farboor, M., & Schoner, B. (2001). Analysis and Synthesis of Palestrina-Style Counterpoint Using Markov Chains. *Proceedings of the 2001 International Computer Music Conference*, S. 471-475.
- Gers, F., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: continual prediction with LSTM. *Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, S. 850-855.
- Hochreiter, S., & Schmidhuber, J. (November 1997). Long Short-Term Memory. *Neural Computation, Volume 9 / Issue 8 /*, S. 1735-1780.
- Holland, S. (2000). Artificial Intelligence in music education: a critical review. *Miranda, E. (ed.) Readings in Music and Artificial Intelligence, Contemporary Music Studies Vol. 20.*, S. 239 ff.
- Huang, C.-Z. A., Cooijmans, T., Roberts, A., Courville, A., & Eck, D. (2017). COUNTERPOINT BY CONVOLUTION. *ICLR 2017 conference*. Toulon: ICLR 2017 conference submissions.
- Kingma, D. P., & Ba, J. L. (2015). Adam: A Method for Stochastic Optimization. *Conference paper, 3rd International Conference for Learning Representations*.
- Liang, F. (2016). *BachBot: Automatic composition in the style of Bach chorales*. Cambridge: Department of Engineering University of Cambridge.
- Manaris, B., Roos, P., Machado, P., Krehbiel, D., Pellicoro, L., & Romero, J. (2007). A corpus-based hybrid approach to music analysis and composition. *Proceedings of the National Conference on Artificial Intelligence (Vol. 22, No. 1)*, S. 839.
- Mann [Übersetzung], A., & Fux, J. J. (1965). *The Study of Counterpoint*. New York: W. W. Norton & Company.
- Matsumoto, M., & Nishimura, T. (Januar 1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation. Volume 8, Issue 1.*, S. 3-30.
- Meehan, J. R. (1980). An Artificial Intelligence Approach to Tonal Music Theory. *Computer Music Journal Vol. 4, No. 2, Intelligence and Music Part 1*, S. 60-65 .

- MIZ, Deutsches Musikinformationszentrum. (März 2019). *Abschlussprüfungen in Studiengängen für Musikberufe*. Abgerufen am 12. November 2019 von MIZ.org:
http://www.miz.org/downloads/statistik/13/13_Abschlusspruefungen_Musikberufe_Studienfach.pdf
- Newcomb, S. R. (1985). LASSO: an intelligent computer-based tutorial in sixteenth-century counterpoint. *Computer Music Journal archive (Volume 9 Issue 4)*, S. 49-61.
- Olah, C. (2015). *Understanding LSTMs*. Von <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> abgerufen
- Russell, S., & Norvig, P. (2012). *Künstliche Intelligenz Ein moderner Ansatz (3. Aufl.)*. München: Pearson Studium.
- Scholzen, M. J. (2019). *CounterPai eine lernunterstützende Software für die Kompositionslehre*. Von https://github.com/Ni2Be/CounterPAI/releases/download/BETA_v0.7.0/Dokumentation.CounterPai.pdf abgerufen
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1), S. 1929-1958.
- Wiil, U. K. (2004). *Computer Music Modeling and Retrieval, Revised Papers*. Esbjerg: Springer.

Abbildungsverzeichnis

Abb. 1 Dialog im Gradus ad Parnassum. Der Schüler Joseph erklärt dem Meister Aloyse, was er sich beim Erstellen seines Stücks gedacht hat.....	8
Abb. 2 Intervalle werden immer einschließlich der untersuchten Noten berechnet. Für die Intervalle zwischen c.f. und c.p.: von C nach G ist das Intervall 5, von D nach F 3. Für die Sprünge in der oberen Stimme: G nach F 1. In der unteren Stimme: C nach D ebenfalls 1.	8
Abb. 3 Verschiedene Arten der Bewegung.	9
Abb. 4 Taktpositionen.	9
Abb. 5 Beatpositionen.	9
Abb. 6 Ein gehaltenes C im zweiten Takt. Die Anfrage Tied(N) würde das Ergebnis „True“ liefern.	9
Abb. 7 Die Anfrage Value(N) würde von links nach rechts die Ergebnisse „halbe Note“, „Viertelnote“, „Achtelnote“, „Achtelnote“ liefern.	9
Abb. 8 Die Tabelle von Regel 2 „Bevorzugte Motion“. Mit welcher Bewegung eine Note erreicht wird, spielt nur auf dem Downbeat eine Rolle, dargestellt durch die Notation PosN = DBeat...	10
Abb. 9 Die Kontrapunktregeln. R5 wurde um eine Teilregel erweitert und hängt jetzt auch von der vorherigen DBeat-Note ab.	10
Abb. 10 CounterPai v0.7.0. Die Informationen rechts beziehen sich auf die umkreiste Note.	11
Abb. 11 CounterPai-Komponentendiagramm.	11
Abb. 12 Komponente Evaluator mit internen Klassen.	12
Abb. 13 Ausschnitt einer sheet-Datei mit angehängten Nachrichten.	12
Abb. 14 Nachricht des AI_Evaluators der annimmt, dass Regel 2 und 4 gebrochen wurden.	12
Abb. 15 Ausschnitt einer Learn_Settings-Datei. Hinter der „data_converter_info“ sind außerdem Angaben über die möglichen „nn_typen“, „optimizer“, etc. gespeichert.	13
Abb. 16 Hauptbildschirm des AI_Config_CLI.	13
Abb. 17 Von Hand geschriebener Kontrapunkt aus dem Gradus ad Parnassum (fig. 85).	15
Abb. 18 Ein aus fig. 88 aus dem Gradus ad Parnassum mutierter Sheet (sheet 0 der Mutationsvaliditätsdaten). Die umkreisten Noten wurden mutiert (Originalnoten schemenhaft abgebildet).	16
Abb. 19 Relative Wahrscheinlichkeiten einer generierten Folgenote. Links: Ausgangsnote, Rechts: mögliche Folgenoten.	17
Abb. 20 Relative Wahrscheinlichkeiten der c.p.-Noten. Alle Eigenschaften sind voneinander unabhängig.	17
Abb. 21 Ein typischer mit dem stochastischen Kontrapunktgenerator erstellter Kontrapunkt (sheet 0 der stochastischen Trainingsdaten).	18
Abb. 22 Vertretene Notenhöhen (Pitch) im cantus firmus der verschiedenen Daten. Aufgrund eines Programmierfehlers wird C6 nur erzeugt, wenn es durch einen Sprung, der über das untere Ende der oberen Stimme hinausgeht, generiert wird. Dieser Fehler wurde erst erkannt, nachdem schon viele Versuche mit den Netzen abgeschlossen waren, und konnte aus Zeitgründen nicht berichtigt werden. Er sollte aber keinen Einfluss auf den Lernerfolg der Netze haben.	19
Abb. 23 Verteilung der Notenwerte im c.p.	20

Abb. 24 Erzeugte Bewegung der Noten. NoMo bezieht sich immer auf die erste Note, da zu ihr keine Bewegung stattfindet.	20
Abb. 25 Die gebrochenen Kontrapunktregeln.	20
Abb. 26 Das sheet-Format. Die Noten sind nacheinander aufgelistet. Die Zusätze sind wie folgt definiert: p: 69 ist die Tonhöhe (pitch) A4 im midi-Format, va: 2 (Notenwert: value) gibt an, dass es sich um eine halbe Note handelt, f: 0, s: 0 bedeutet, dass die Note weder vermindert (flat) noch erhöht (sharp) ist, t: 0 bedeutet, dass die Note nicht gehalten (tied) ist.	22
Abb. 27 Regeln des 5.Sp. Counterpoint	22
Abb. 28 Regeln des 1.Sp. Counterpoint	22
Abb. 29 Die Noten aus Abb. 27 dargestellt als Achtelnoten. Blau umrandet der Bereich, der zur Bewertung aller Kontrapunktregeln benötigt wird. Rot umrandet der Bereich, der benötigt wird, um bewerten zu können, ob das C richtig aufgelöst wurde.	22
Abb. 30 Ausschnitt der binären Codierung. Die Taktstriche sind nur zur Orientierung eingetragen und nicht Teil des Codes.	23
Abb. 31 Die Elemente werden hier vereinfacht durch ganze Zahlen dargestellt. In diesem Beispiel wird von einigen Elementen Regel R3 verletzt (In den n-Tupeln durch das tiefgestellte R3 angedeutet).	24
Abb. 32 Ausschnitt eines Kontrapunkts, dargestellt als Notenblatt und formatierter Code. Die Matrix, die für die Prüfung auf Ähnlichkeit generiert würde, ist blau umrandet und bezieht sich auf die mit dem Pfeil versehene Note.	25
Abb. 33 Oben: Originalmatrix mit 663 Elementen. Mitte: Pair-Code mit 49 Pärchen. Unten: 64bit-Code mit 11 64bit-Integern.	26
Abb. 34 Ein Performance-Test, bei dem die postseparierten Trainingsdaten mit den Test- und Validationsdaten verglichen wurden, zeigt den Vorteil der 704bit-Lösung. Verglichen mit der 64bit-codierten Lösung entsteht ein Zeitersparnis von 96,34%.	26
Abb. 35 Die Daten aus Abb. 32, blau umrandet die Matrix für 17 Achtelscheiben, orange für 9 und grün für 5.	27
Abb. 36 stoch: stochastische Daten, post: postseparierte Mutationsdaten, prä: präseparierte Mutationsdaten, train: Trainingsdaten, test: Testdaten, valid: Validationsdaten	27
Abb. 37 Feedforward Netz mit einem Hidden Layer.	28
Abb. 38 σx (blau) und $\sigma' x$ (rot). Bei großen Abweichungen sind die Gradienten am kleinsten.	28
Abb. 39 Schematische Darstellung eines LSTM. xt beschreibt den Input einer Sequenz an der Position t (zum Beispiel eine Achtelnotenscheibe). ht beschreibt den jeweiligen Output zur Sequenz an der Position t (die Netze in dieser Arbeit verwerfen alle Outputs außer den letzten). 1: State der Memory-Cell, 2: Forget Gate, 3: Speichern neuer Informationen, 4: Output-Gate. Quelle: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (03.12.2019) (Nummerierung nachträglich eingefügt).	29
Abb. 40 Anzahl der Kanten zwischen Input und Hidden Layer mit 312 Neuronen. Feedforward (blau), LSTM (rot) y: Anzahl Kanten, x: Betrachtete Achtelnotenscheiben.	30
Abb. 41 Ursprüngliche Noten. Betrachtete Note umkreist.	31
Abb. 42 Die Noten aus Abb. 41 dargestellt als Achtelnoten. Betrachtete Note umkreist.	31
Abb. 43 Die Noten aus Abb. 41 im codierten Format als Achtelnotenscheiben. Betrachtete Note mit Pfeil versehen.	31

Abb. 44 Der F1-Score der mit den post- und präseparierten Daten trainierten neuronalen Netze im direkten Vergleich. Die Farbcodierung bezieht sich immer auf eine Zeile (8 Werte). Rechts die besten Ergebnisse der jeweiligen Versuchsreihen. Das Netzdiagramm zeigt die F1-Scores, das perfekte Netz würde alle 9 Ecken im Punkt 1,000 schneiden. *Für R7 kann keine Aussage getroffen werden, da diese Regel in den post- und präseparierten Validationsdaten nicht verletzt wird.	34
Abb. 45 Der F1-Score der mit den prä- und stochastischen Daten trainierten neuronalen Netze im direkten Vergleich. Die Farbcodierung bezieht sich immer auf eine Zeile (8 Werte). Rechts die besten Ergebnisse der jeweiligen Versuchsreihen. Das Netzdiagramm zeigt die F1-Scores. *Für R7 kann keine Aussage getroffen werden, da diese Regel in den prä- und postseparierten Validationsdaten nicht verletzt wird.....	36
Abb. 46 Der F1-Score der mit den stochastischen Daten trainierten neuronalen Netze im direkten Vergleich. Abgebildet sind die in der Summe ihrer F1-Scores am besten abschneidenden Netze. Die Farbcodierung bezieht sich immer auf eine Zeile (8 Werte). Rechts die besten Ergebnisse der jeweiligen Versuchsreihen. Das Netzdiagramm zeigt die F1-Scores. *Für R7 kann keine Aussage getroffen werden, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.	38
Abb. 47 Nach der ersten Auswahl sind in den 1.000 Ausgewählten sheets alle Regeln über 1.000-mal gebrochen außer R4 und R5. Gegen R8 wird 1.091-mal verstoßen, da in einigen Kontrapunkten mehrere Noten gegen sie verstoßen.....	40
Abb. 48 Nach der 2. Auswahl sind in insgesamt 1.233 sheets alle Regeln über 1.000-mal gebrochen.	40
Abb. 49 Die Verteilung der c.f.-Noten in den ausgewählten Daten ähnelt einer Sinuskurve mit einem Sprung in der Mitte, ergibt sich aber aus den Wahrscheinlichkeiten, mit denen der c.f.-Generator arbeitet.	41
Abb. 50 Der F1-Score der mit den stochastischen und selektierten Daten trainierten neuronalen Netze im direkten Vergleich. Abgebildet sind die in ihrer Summe der F1-Scores am besten abschneidenden Netze. Die Farbcodierung bezieht sich immer auf eine Zeile (8 Werte). Rechts die besten Ergebnisse der jeweiligen Versuchsreihen. Das Netzdiagramm zeigt die F1-Scores. *Für R7 kann keine Aussage getroffen werden, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.	42
Abb. 51 Der F1-Score einzelner Netze der Versuchsreihen A3 und A5 im direkten Vergleich. Abgebildet sind die in der Summe ihrer F1-Scores am besten abschneidenden Netze. Die Farbcodierung bezieht sich immer auf eine Zeile und Versuchsreihe (2 Werte). Die Netzdiagramme zeigen die F1-Scores. A3 und A5 beziehen sich jeweils auf die Versuchsreihen A3 Versuchsreihe „Training mit präseparierten Mutationsdaten“ und A5 Versuchsreihe „Training einzelner Netze auf alle Regeln“. *Für R7 kann keine Aussage getroffen werden, da diese Regel in den jeweiligen Validationsdaten nicht verletzt wird.....	44
Abb. 52 Der F1-Score eines Einzelversuchs, bei dem Netze auf Regel R4 trainiert wurden, der Achtelnotenscheibenbereich aber von 17 auf 49 erhöht wurde. Die Farbcodierung bezieht sich immer auf eine Zeile (2 bzw. 4 Werte). Das Netzdiagramm zeigt die F1-Scores. Trainingsdaten: stochastische Trainingsdaten, Validationsdaten: postseparierte Validationsdaten. Netzeinstellungen: FF wie in A2, DL3sr/ar wie in A5. *Für R7 kann keine Aussage getroffen werden, da diese Regel	

in den jeweiligen Validationsdaten nicht verletzt wird. **Für die Einzelregeln wurden nur Netze trainiert, die R4 bewerten können.	45
Abb. 53 Einstellungen der Netze. Für die ar-Netze ändert sich nur die Anzahl der Output-Neuronen auf 9. i: Input-Neuronen, h: hidden Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainierte.....	46
Abb. 54 Farbverlauf jeweils für alle F1-Scores (24 Werte), Acc (24 Werte) und die jeweiligen Summen (3 Werte). Die Summe soll nur einen Anhaltspunkt geben, wie die Netze im Vergleich zueinander abgeschnitten haben, sie eignet sich nicht zur qualitativen Bewertung.	66
Abb. 55 Confusion Matrix: postseparierte Daten. Farbverlauf jeweils pro Regel und Viertel der Matrix (3 Werte) *Für Regel R7 ist kein Farbverlauf eingetragen, da diese Regel in den postseparierten Daten nicht verletzt wird.	67
Abb. 56 Einstellungen der Netze. i: Input-Neuronen, h: Hidden-Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainiert.	67
Abb. 57 Farbverlauf jeweils für alle F1-Scores (24 Werte), Acc (24 Werte) und die jeweiligen Summen (3 Werte). Die Summe soll nur einen Anhaltspunkt geben, wie die Netze im Vergleich zueinander abgeschnitten haben, sie eignet sich nicht zur qualitativen Bewertung.	69
Abb. 58 Confusion Matrix: postseparierte Daten. Farbverlauf jeweils pro Regel und Viertel der Matrix (3 Werte).....	70
Abb. 59 Einstellungen der Netze. i: Input-Neuronen, h: Hidden-Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainiert. (Einstellungen der Versuchsreihen A2-A4 sind identisch)	70
Abb. 60 Farbverlauf jeweils für alle F1-Scores (24 Werte), Acc (24 Werte) und die jeweiligen Summen (3 Werte). Die Summe soll nur einen Anhaltspunkt geben, wie die Netze im Vergleich zueinander abgeschnitten haben, sie eignet sich nicht zur qualitativen Bewertung.	71
Abb. 61 Confusion Matrix: postseparierte Daten. Farbverlauf jeweils pro Regel und Viertel der Matrix (3 Werte).....	72
Abb. 62 Einstellungen der Netze. i: Input-Neuronen, h: Hidden Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainiert. (Einstellungen der Versuchsreihen A2-A4 sind identisch)	72
Abb. 63 Wiederholung (DL WH) des Trainings für Regel R5.	73
Abb. 64 Die Confusion Matrix dargestellt über den Verlauf von 100 Epochen.	73
Abb. 65 Farbverlauf jeweils für alle F1-Scores (80 Werte) und die jeweiligen Summen (10 Werte). **Für Regel R7 ist kein F1-Score und angegeben, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.....	74
Abb. 66 Farbverlauf jeweils für alle Acc (80 Werte) und die jeweiligen Summen (10 Werte). *Diese Werte sind in den Netzdiagrammen nicht abgebildet. **Für Regel R7 ist keine Acc angegeben, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.	75
Abb. 67 Einstellungen der Netze. DL4, ZL4 und FF2 können als die Standardeinstellungen angesehen werden, die fettgedruckten Werte geben jeweils die Abweichung zu diesen	

Einstellungen an. i: Input-Neuronen, h: hidden Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainiert.	75
Abb. 68 Die in den Versuchen bestabschneidenden Netze.	76
Abb. 69 Confusion Matrix: postseparierte Daten. Farbverlauf jeweils pro Regel und Viertel der Matrix (4 Werte) *Für Regel R7 ist kein Farbverlauf eingetragen, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.	76
Abb. 70 Gebrochene Regeln nach Auswahl von R8.	78
Abb. 71 Gebrochene Regeln nach der 1. Iteration.	78
Abb. 72 Die Elemente der selektierten Daten, die auch in den postseparierten Validationsdaten auftauchen, und wie oft diese Elemente in diesen postseparierten Validationsdaten vorhanden sind.	79
Abb. 73 Farbverlauf jeweils für alle F1-Scores (48 Werte) bzw. die Acc (48 Werte) und die jeweiligen Summen (6 Werte). *Für Regel R7 ist kein F1-Score und keine Acc angegeben, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.	79
Abb. 74 Einstellungen der Netze. *Die Versuche wurden mit 100 Epochen wiederholt, es konnten aber keine besseren Ergebnisse erzielt werden. Für die 100 Epochen-Netze wurden keine Zwischenergebnisse gesammelt, deshalb werden hier nur die 30 Epochen-Netze besprochen. i: Input-Neuronen, h: hidden Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainierte.	80
Abb. 75 Confusion Matrix: postseparierte Daten. Farbverlauf jeweils pro Regel und Viertel der Matrix (6 Werte). *Für Regel R7 ist kein Farbverlauf eingetragen, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.	80
Abb. 76 Die Wahrscheinlichkeit für das Intervall der betrachteten Note unter der Bedingung, dass es keine Vorgängernote gibt.	83
Abb. 77 Es sollte so oft wie möglich versucht werden, CoMo und ObMo zu erreichen.	83
Abb. 78 Unter der Voraussetzung, dass die Bedingung von R1 nicht zutrifft, also die Note nicht die erste oder letzte Note ist, sollten dissonant intervals bevorzugt werden.	84
Abb. 79 Tied Notes dürfen nur verwendet werden, wenn sie auf einem DBeat stehen. Sie dürfen außerdem dissonant sein, wenn sie richtig aufgelöst werden. Werden sie nicht richtig aufgelöst, scheiden die dissonanten Intervalle aus. St : <i>Skip Interval</i>	84
Abb. 80 Für die vorletzte Note gibt es nur je eine Möglichkeit je nachdem, ob der c.f. im Bass oder Soprano steht.	85
Abb. 81 Steht der c.f. im bass, ist das Intervall der vorletzten Note immer vorgegeben.	85
Abb. 82 Es ergibt sich, dass zu einem perfect Interval nur in Contrary Motion geschritten werden darf und zu einem imperfect Interval in jeder der Motions. It ist für diese Regel also unabhängig von It – 1.	86
Abb. 83 Vereinfachte Motion-abhängige Intervals.	86
Abb. 84 Notensprünge größer als M3 sind grundsätzlich verboten (Tri und M6 sind in s>M3 enthalten). Ausnahmen bildet die m6 und, falls nicht von der P1 gesprungen wird, die P5 und P8.	87
Abb. 85 Tied Notes sind nur auf dem DowBeat erlaubt.	87

Abb. 86 Alle nicht aufgeführten Werte ergeben sich zu 1.0.	88
Abb. 87 Verbotene Achtelnoten.	88
Abb. 88 Für die erste Note liefert R1 drei Noten mit der Wahrscheinlichkeit 1.0 (grün – C4, C5, C6) und zwei mit der Wahrscheinlichkeit 0.5 (gelb – G4, G5).	89
Abb. 91 R2 Bevorzugte Motion	89
Abb. 91 R3 Bevorzugte Intervalle.	89
Abb. 91 R2 und R3.	89
Abb. 93 R5 Bevorzugte Motion-abhängige Intervals.	90
Abb. 93 R2, R3 und R5.	90
Abb. 95 R6 Verbotene Skips.	90
Abb. 95 R2, R3, R5 und R6.	90
Abb. 96 Vollständiges Klassendiagramm mit Komponentengrenzen. AI Evaluator ist Teil der Folgearbeit und zu diesem Zeitpunkt noch nicht implementiert.	91
Abb. 97 Klassendiagramm der Komponente Sheet_To_Audio.	92
Abb. 98 Ausschnitt einer sheet-Datei. (p: Pitch, va: Notenwert, vo: Voice, f: Flat, s: Sharp, t: Tied)	92
Abb. 99 Der Funktion add_note(...) wird als Input ein C5 als Viertelnote und die Distanz 26 Sechzehntel übergeben. Die ganze Note B4 wird zu einer halben verbunden mit einer Viertelnote und die neue Note wird eingefügt.	93
Abb. 100 Oben ein Midi-File angezeigt als Zeitleiste. Man kann in der Basslinie erkennen, dass die Töne nicht durchgängig spielen. Unten das Ergebnis des Imports.	95
Abb. 101 Klassendiagramm der Komponente GUI.	96
Abb. 102 Die Komponente Utility	97
Abb. 103 Die Komponente Evaluator	98
Abb. 104 Eine Note im sheet-Format mit angefügten Nachrichten. (NOTE_EVALUATION: p: Bar-Position, b: Beat-Position, j: Jump-Interval, d: Direction, m: Motion, i: Interval mit dem c.f. RULE_EVALUATION: prb: probability, R2, R3: Regeln 2 und 3 wurden verletzt.	98

Externe Quellen:

Abb. 37 erstellt mit NN-SVG <http://alexlenail.me/NN-SVG/index.html> (03.12.2019)

Abb. 38 und Abb. 40 erstellt mit dem Online-Funktionsplotter von <https://www.mathe-fa.de/> (03.12.2019)

Abb. 39 Quelle: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (03.12.2019)
(Nummerierung nachträglich eingefügt)

Glossar

CP	Counterpoint (Die Technik)	FBar	First-Bar
c.p.	Counterpoint (Die Melodie). Vom Studenten eingegebene Melodie.	MBar	Mid-Bar
c.f	Cantus Firmus. Vorgegebene Melodie.	BLBar	Before-Last-Bar
CoMo	Contrary Motion	LBar	Last-Bar
ObMo	Oblique Motion	DBeat	Down-Beat
DiMo	Direct Motion	UBeat	Up-Beat
I	Interval	WBeat	Weak-Beat. WBeat1 ist der erste, WBeat2 der zweite im Takt
PerCI	Perfect consonant interval (P1, P5, P8)	N	Note
iPerCI	Imperfect consonant interval (m/M3, m/M3)	$N_t \neq \emptyset$	Gibt an, dass eine Note nicht vorhanden ist (z.B. würde $N_{t-1} \neq \emptyset$ bedeuten, dass N_t die erste Note ist).
CI	Consonant interval: Alle PerCI und IPerCI	Tied(N)	Ergibt true, wenn die Note mit einem Haltebogen mit der davorliegenden Note verbunden ist.
DisI	Dissonant interval (m/M2, P4, m/M7)	Pos(N)	Gibt die Beat-Position (z.B. DBeat) der Note zurück.
S	Das Sprungintervall einer Note. (z.B. C4 nach D4: M2)	Voice(c.f.)	Gibt die Stimme (Soprano oder Bass) des c.f. zurück.

Nomenklatur

ZL	Ein LSTM-Netz, das mit dem Input aus 6.1.3 Zusammenlaufender sequenzieller Input trainiert wurde.	st / stoch / stochastisch	Die stochastischen Daten (Trainings-, Test-, Validationsdaten). Beschrieben in 4 Trainingsdaten. Auch als Zusatz für die Netze vgl. pr
DL	Ein LSTM-Netz, das mit dem Input aus 6.1.2 Durchlaufender sequenzieller Input trainiert wurde.	sel / selektiert	Die selektierten Trainingsdaten. Beschrieben in A6 Versuchsreihe „Reduzieren der Trainingsbeispiele“. Auch als Zusatz für die Netze vgl. pr
FF	Ein Feedforward-Netz, das mit dem Input aus 6.1.1 Feedforward-Input trainiert wurde.		
pr / prä / präsepariert	Die präseparierten Mutationsdaten (Trainings-, Test-, Validationsdaten). Beschrieben in 4 Trainingsdaten. Auch als Zusatz für die Netze z.B. „DLpr“: ein DL-Netz, das mit dem präseparierten Trainingsdaten (falls nicht anders notiert) trainiert wurde.	sr / Einzelregel	Zusatz der Netze mit der Bedeutung, dass das Netz auf eine einzelne Regel trainiert wurde (Single Rule). Netze in A2-A4 und mit sr gekennzeichnete Netze in A6. Z.B. „DLselsr“: DL-Netz, das auf den selektierten Daten auf eine einzelne Regel trainiert wurde. (Wird nur angegeben, wenn es aus dem Kontext nicht ersichtlich ist.)
po / post / postsepariert	Die postseparierten Mutationsdaten (Trainings-, Test-, Validationsdaten). Beschrieben in 4 Trainingsdaten. Auch als Zusatz für die Netze vgl. pr	ar / Alle Regeln	Zusatz der Netze mit der Bedeutung, dass das Netz auf alle Regeln trainiert wurde (All Rules). Netze in A5 und mit ar gekennzeichnete Netze in A6. Beispiel vgl. sr

Anhang

A Ergebnisse

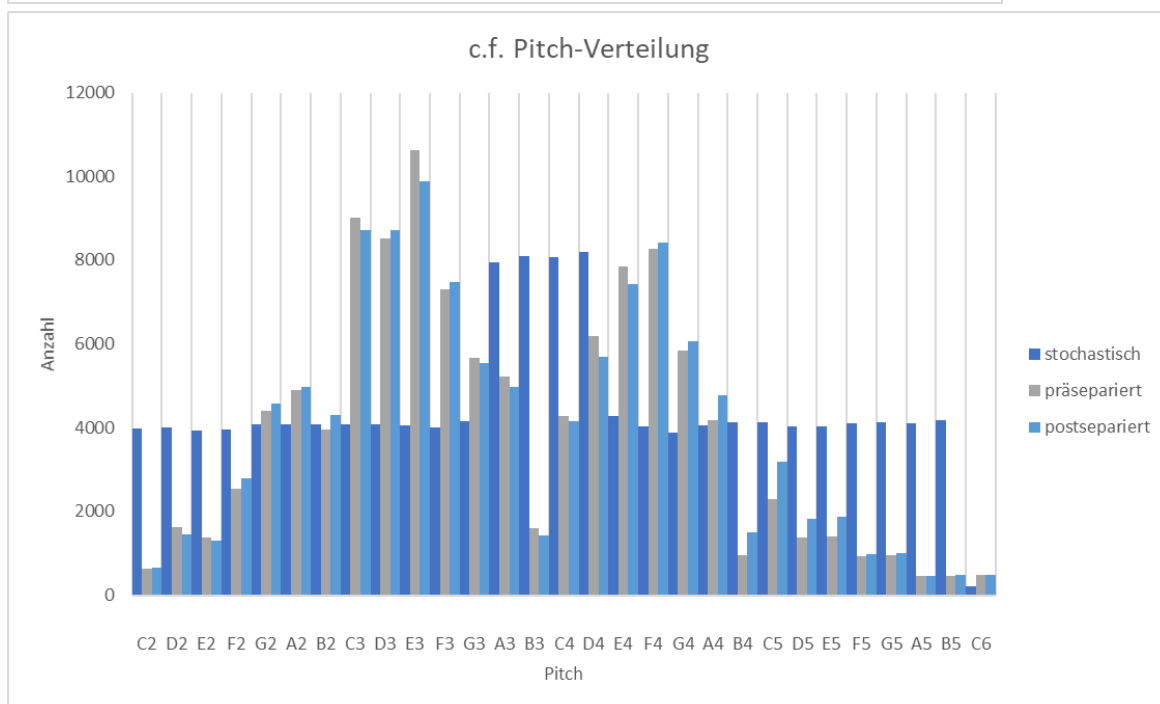
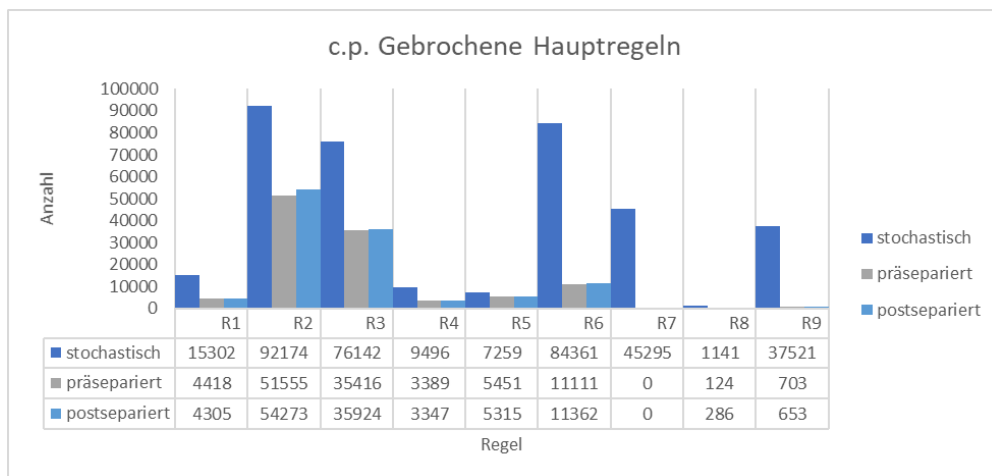
A1 Vollständige Analyse der generierten Daten

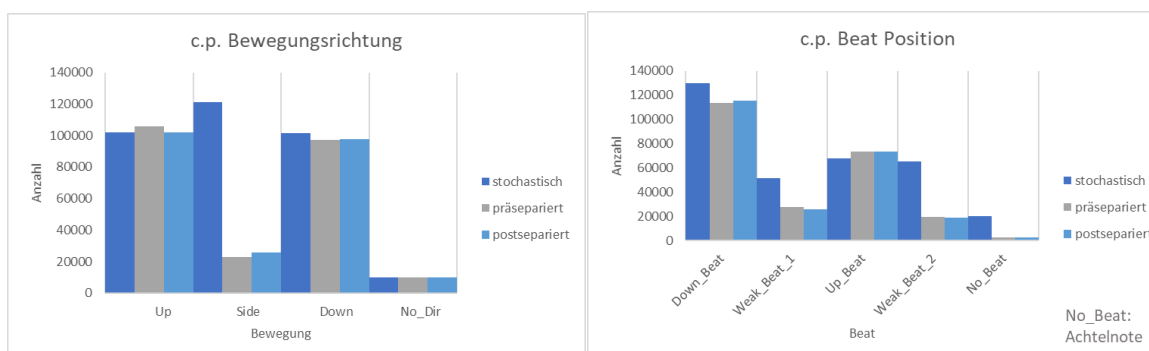
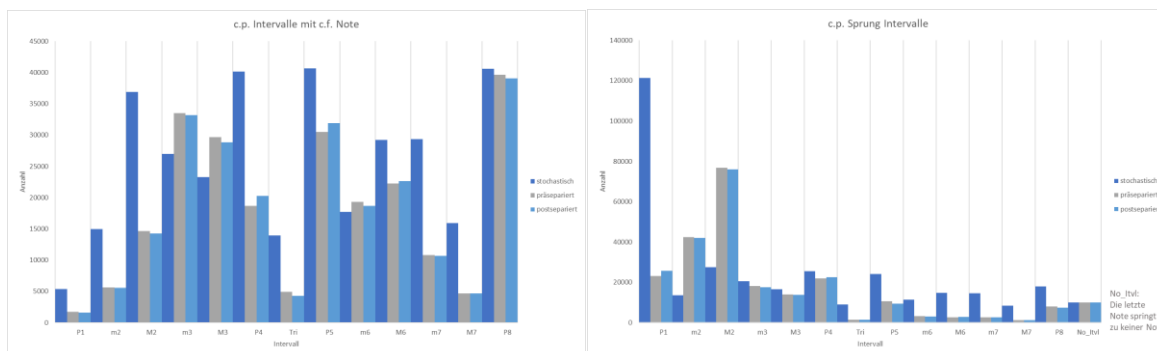
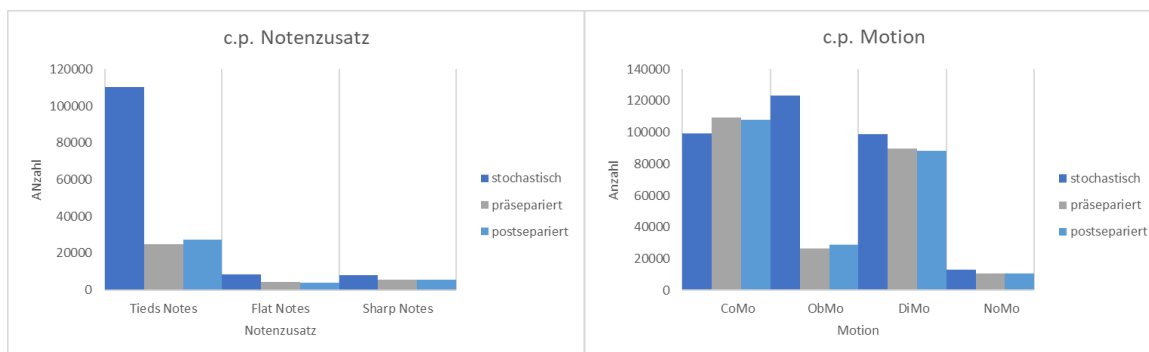
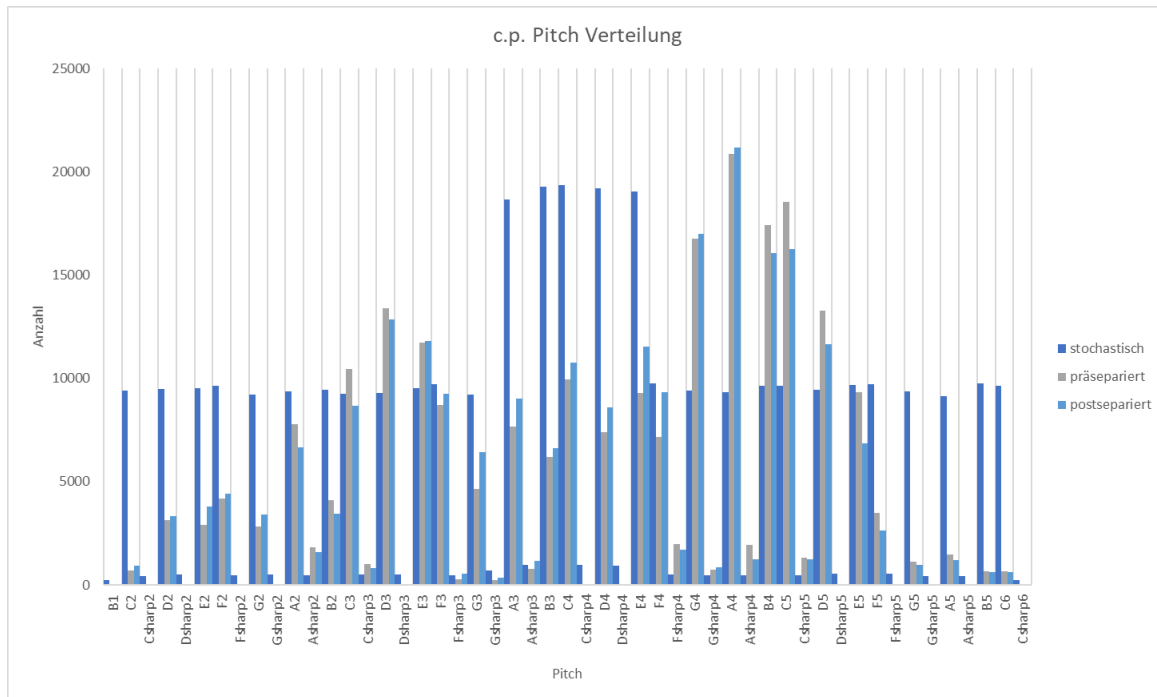
Die Daten wurden auf alle Eigenschaften, die in Kapitel „Die Kontrapunktregeln“, S. 8 beschrieben wurden, untersucht, um auf Unregelmäßigkeiten zu prüfen, die den Lernerfolg beeinträchtigen könnten.

A1.1 Trainingsdaten

Jeweils 10.000 Kontrapunkte.

	stochastisch	präsepariert	postsepariert
Elemente	335006	235628	235953

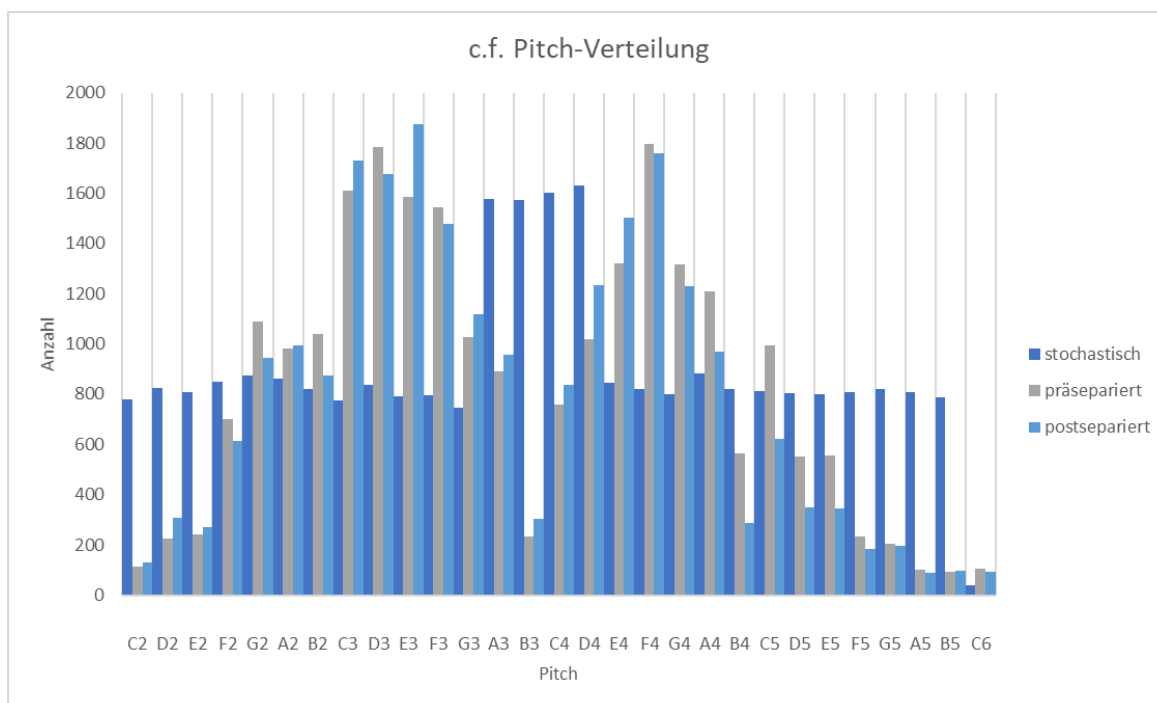
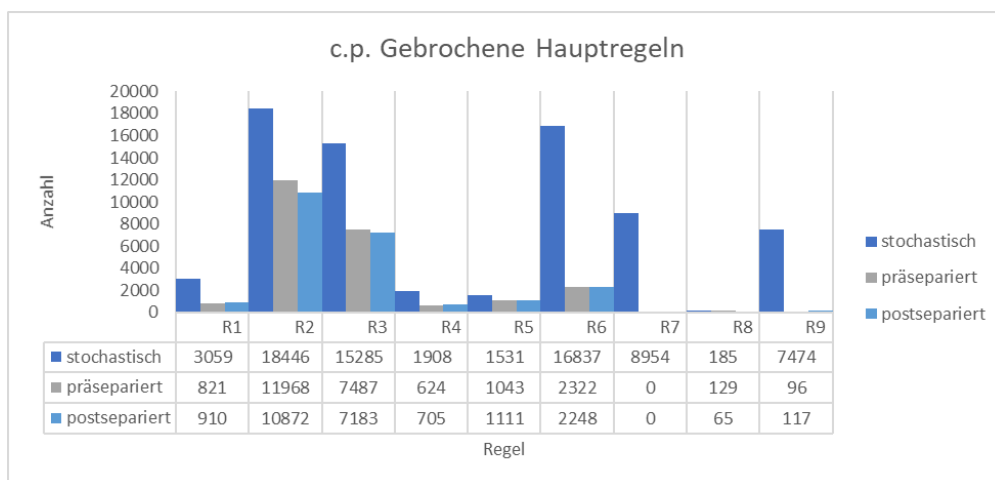


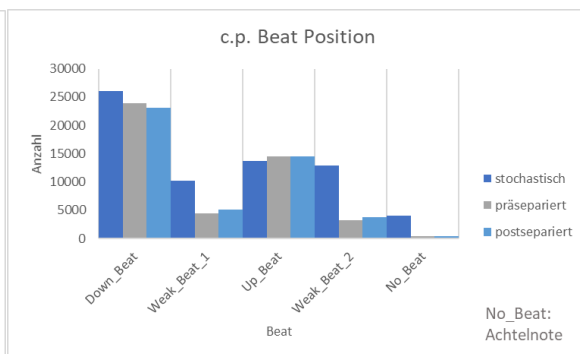
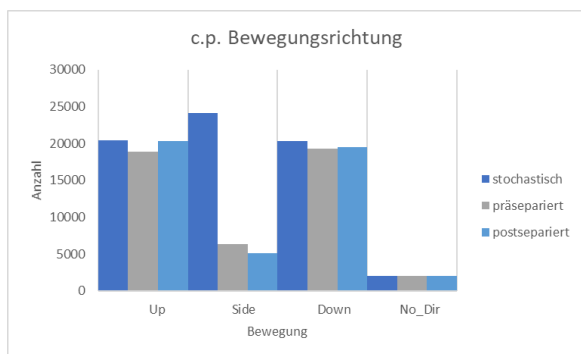
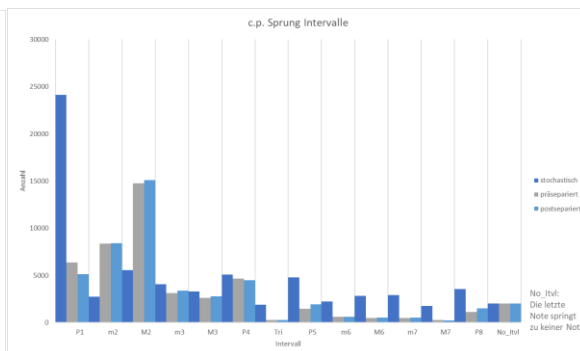
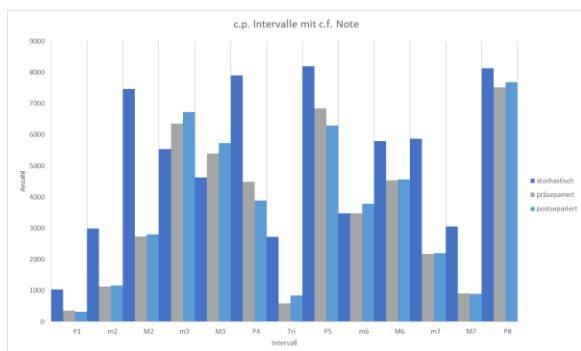
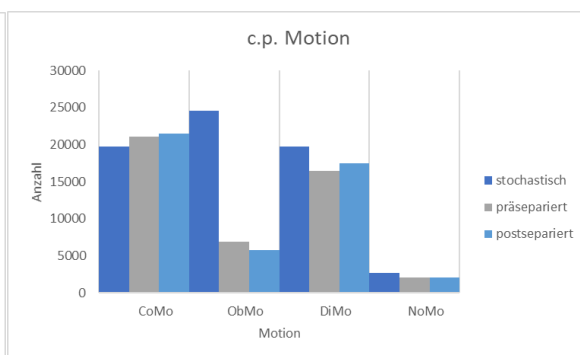
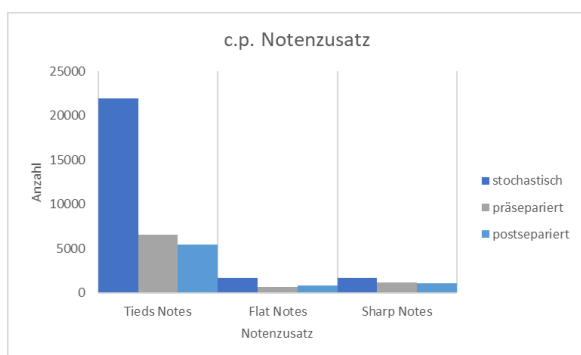
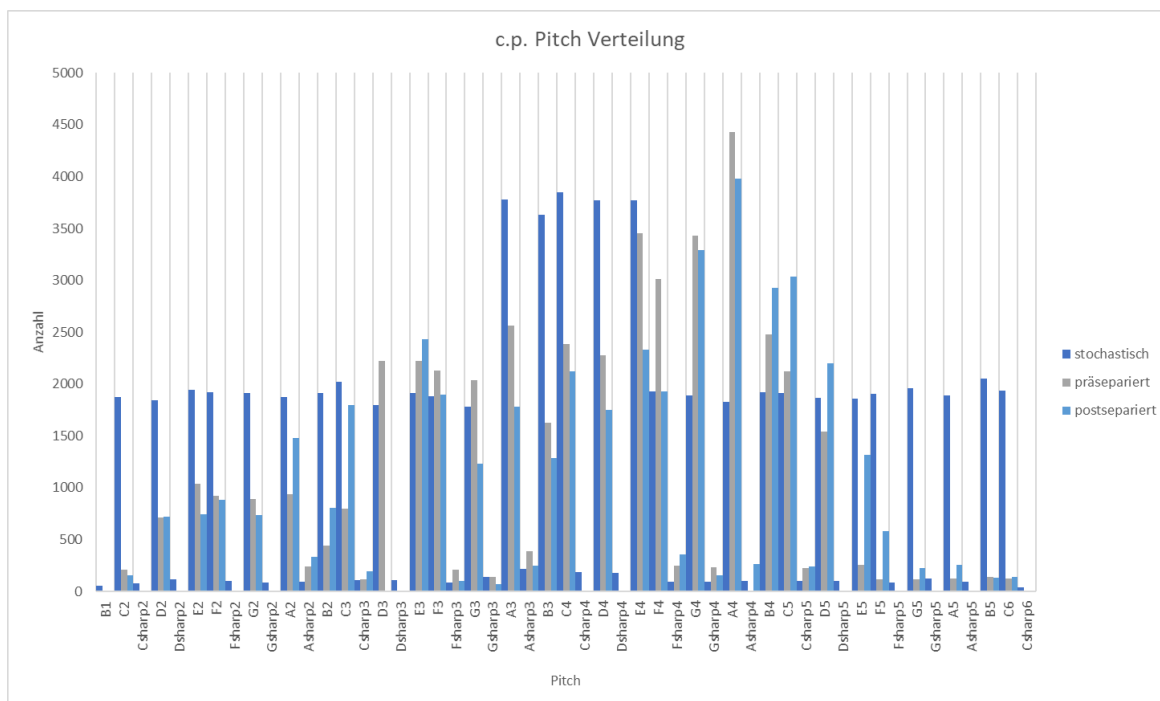


A1.2 Testdaten

Jeweils 2.000 Kontrapunkte.

	stochastisch	präsepariert	postsepariert
Elemente	66834	46883	46535

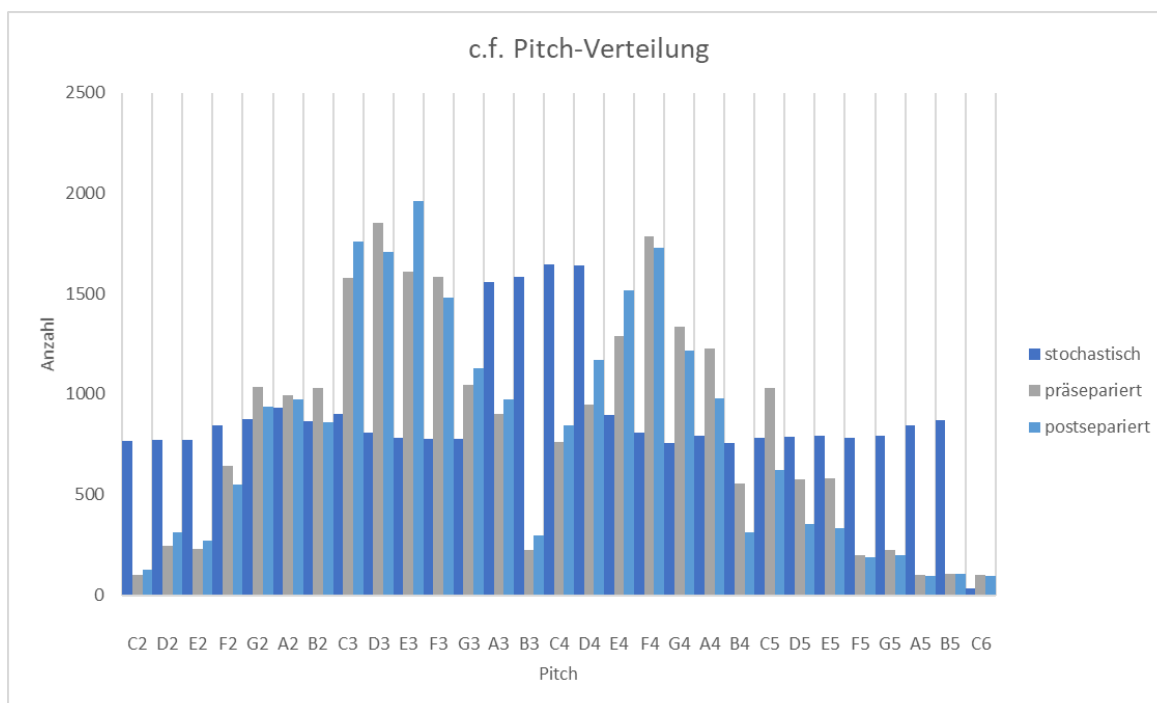
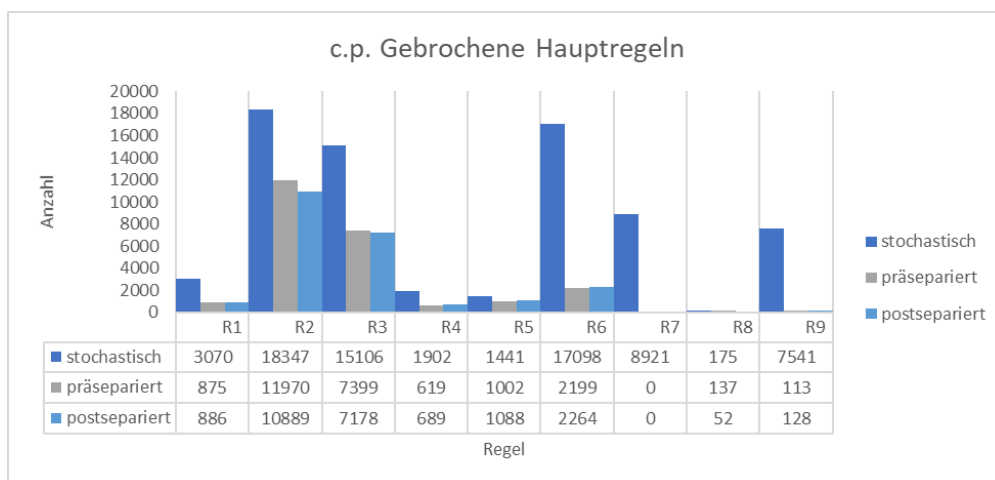


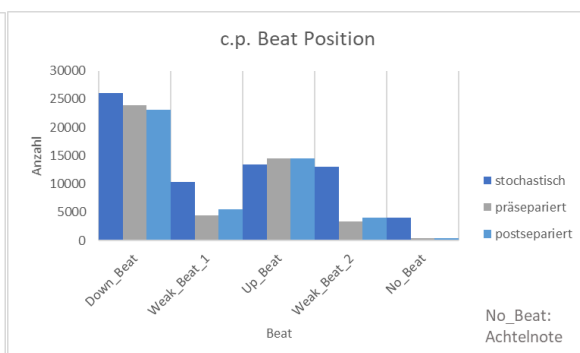
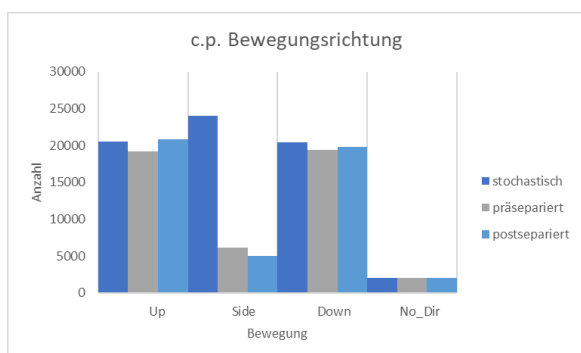
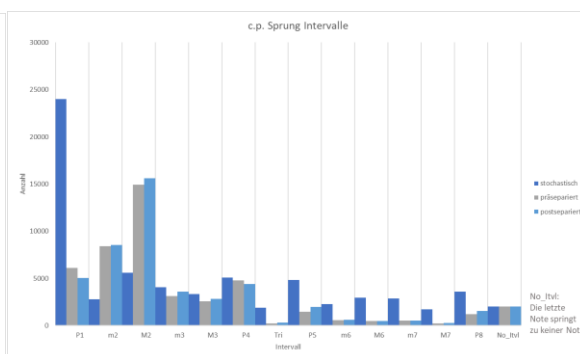
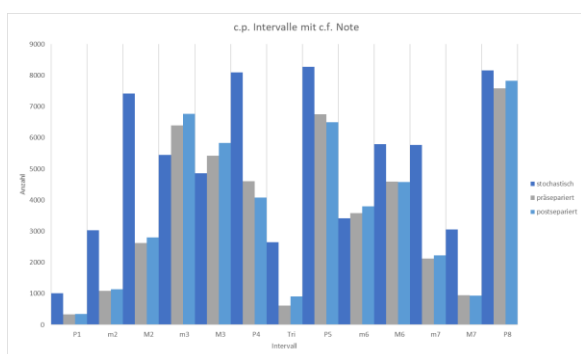
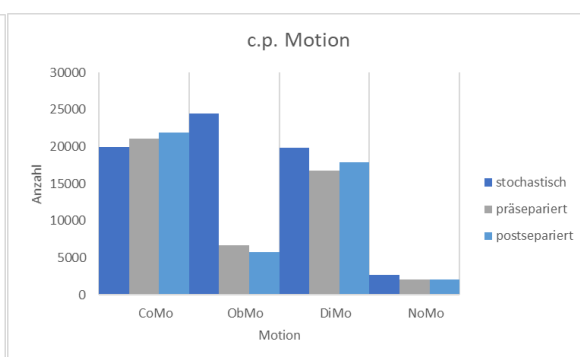
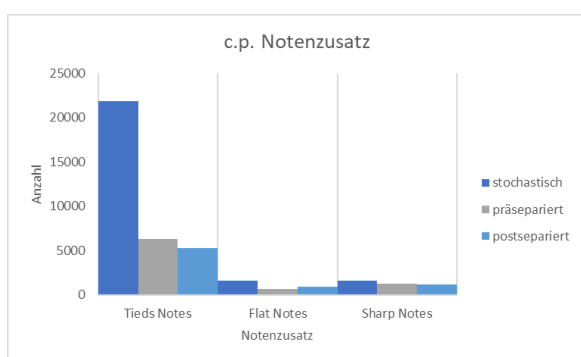
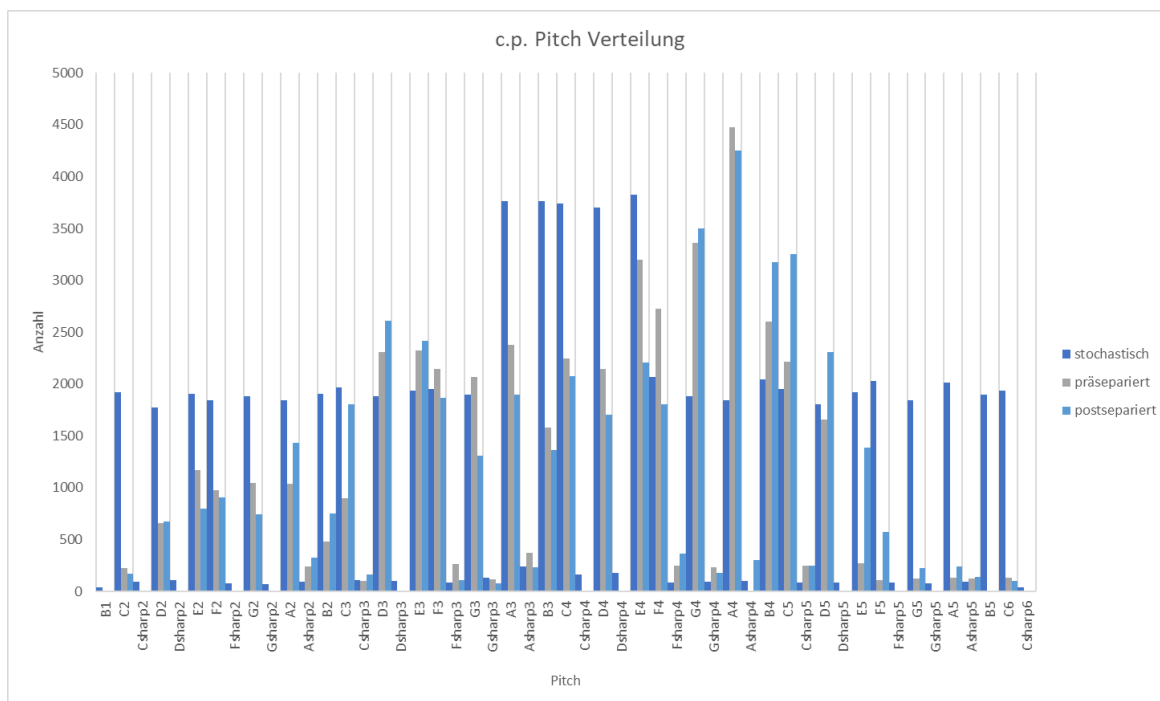


A1.3 Validationsdaten

Jeweils 2.000 Kontrapunkte.

	stochastisch	präsepariert	postsepariert
Elemente	66970	47702	46635





A1.4 Analyse auf gleiche Datensätze

Die Werte der Tabellen geben an, wie oft ein Element, das in den Trainingsdaten mindestens einmal vorkommt, in den Test- bzw. Validationsdaten vorkommt. (Beschrieben in 4.5.1 Gleiche Datensätze, S. 24)

$ d_b^{Rx} $ für Elemente aus 17 Achtelnotenscheiben										
a	b	R1	R2	R3	R4	R5	R6	R7	R8	R9
stoch train	post test	17	25	1	0	7	9	0	0	0
	post valid	12	26	2	0	6	4	0	0	0
post train	post test	502	6438	3894	352	551	578	0	41	17
	post valid	480	6454	3875	313	548	558	0	30	13
prä train	prä test	145	1600	1070	88	197	181	0	1	0
	prä valid	154	1586	1074	98	180	146	0	2	0

$ d_b^{Rx} $ für Elemente aus 9 Achtelnotenscheiben										
a	b	R1	R2	R3	R4	R5	R6	R7	R8	R9
stoch train	post test	234	455	338	3	15	79	0	2	0
	post valid	195	420	360	5	14	75	0	2	0
post train	post test	641	8612	5353	444	776	924	0	52	27
	post valid	624	8671	5311	435	768	932	0	40	24
prä train	prä test	306	3375	2501	139	379	383	0	1	0
	prä valid	336	3381	2488	141	335	355	0	4	0

$ d_b^{Rx} $ für Elemente aus 5 Achtelnotenscheiben										
a	b	R1	R2	R3	R4	R5	R6	R7	R8	R9
stoch train	post test	292	2312	1743	31	87	530	0	46	0
	post valid	251	2176	1667	32	86	515	0	35	0
post train	post test	712	9315	5844	516	867	1218	0	57	41
	post valid	708	9365	5809	505	844	1212	0	46	41
prä train	prä test	437	5027	3768	169	481	682	0	17	1
	prä valid	481	5096	3769	170	438	667	0	24	0

A2 Versuchsreihe „Training mit postseparierten Mutationsdaten“

Versuchsbeschreibung:

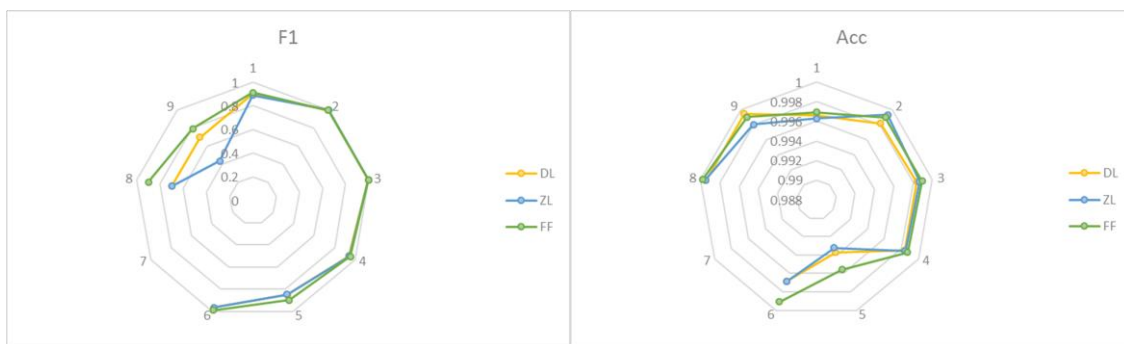
Die DL-, ZL- und FF-Netze⁴⁸ sollen mit den postseparierten Mutationsdaten trainiert werden, um herauszufinden, ob ein Lernerfolg mit Daten stattfinden kann, die von Menschenhand erstellten Daten nahekomen. Dabei soll jeweils pro Regel ein Netz trainiert werden, das die jeweilige Regel erkennen soll.

Trainingsdaten: postseparierte Mutationstrainingsdaten

Testdaten: postseparierte Mutationstestdaten

Validationsdaten: postseparierte Mutationsvalidationsdaten

Ergebnisse:



Die mit den postseparierten Daten trainierten Netze konnten gute Ergebnisse erzielen (vgl. Abb. 54, Abb. 55). Die LSTM-basierten Netze konnten bei den Regeln R8 und R9 nicht so gut abschneiden wie das FF-Netz.

Vor allem das ZL-Netz scheint Probleme zu haben, R9 zu erlernen. R9 „Achtelnoten nur auf *Weak Beats*“:

$$P(N|Value(N) \wedge Pos(N) \wedge (S_t \wedge S_{t+1}))$$

hängt (neben dem Notenwert und dem Sprungintervall) von der Notenposition ab – einer Eigenschaft, die in den LSTM-Netzen voraussetzt, sich daran „erinnern“ zu können, an welcher Stelle die c.f. Noten stehen. Das ZL-Netz wird

	F1			Acc		
	DL	ZL	FF	DL	ZL	FF
R1	0.900	0.890	0.911	0.997	0.996	0.997
R2	0.996	0.998	0.998	0.998	0.999	0.999
R3	0.995	0.996	0.997	0.998	0.999	0.999
R4	0.942	0.946	0.954	0.998	0.998	0.999
R5	0.853	0.847	0.899	0.994	0.993	0.996
R6	0.966	0.968	0.990	0.997	0.997	0.999
R7*						
R8	0.698	0.699	0.900	0.999	0.999	1.000
R9	0.699	0.435	0.791	0.999	0.998	0.999
Σ	7.049	6.779	7.441	7.981	7.980	7.987

Abb. 54 Farbverlauf jeweils für alle F1-Scores (24 Werte), Acc (24 Werte) und die jeweiligen Summen (3 Werte). Die Summe soll nur einen Anhaltspunkt geben, wie die Netze im Vergleich zueinander abgeschnitten haben, sie eignet sich nicht zur qualitativen Bewertung.

*Für Regel R7 sind kein F1-Score und keine Acc angegeben, da diese Regel in den postseparierten Daten nicht verletzt wird.

⁴⁸ DL: LSTM mit durchlaufendem Input | ZL: LSTM mit zulaufendem Input | FF: Feedforward-Netz

eventuell durch das gleichzeitige Annähern von zwei Seiten überfordert. R9 wird in den Trainingsdaten 653-mal gebrochen – vielleicht ist bei dieser Menge kein Lernerfolg möglich.

Die Netze wurden mit den Einstellungen aus Abb. 56 trainiert.

Auswertung:

Die mit den postseparierten Daten trainierten Netze konnten gute Ergebnisse erzielen. Vor allem das FF-Netz macht bei der Bewertung der Regeln R2, R3 und R6 fast keine Fehler. Da für diese Regeln aber verhältnismäßig viele Negativbeispiele in den Daten vorhanden sind, ist nicht auszuschließen, dass nur bereits in den Trainingsdaten gesehene Beispiele wiedererkannt werden.

R2 wird in den postseparierten Daten 10.889-mal gebrochen. Für sie sind, selbst für den 17 Achtelnotenscheiben umfassenden Bereich, 6.459 Elemente in den Trainingsdaten vorhanden, die auch in den Validationsdaten vorkommen. Diese könnten also einfach wiedererkannt werden.

Da R2 „Bevorzugte Motion“:

$$P(M|Pos(N) = DBeat)$$

aber nur von dem unmittelbaren Bereich um die Note abhängt, ist die Analyse auf den Bereich mit 5 Achtelscheiben wohl zutreffender. Hier sind in den Validationsdaten 9.365 Elemente, die R2 brechen, vorhanden, die auch in den Trainingsdaten vorhanden sind. Um R2 bestimmen zu können, reichen zwei Achtelscheiben aus: die der betrachteten Note, um zu prüfen, ob sie auf dem **DBeat** ist, und die eine Achtelnote vor ihr, um die Motion zu bestimmen.

Mit diesen Informationen ist anzunehmen, dass die guten Ergebnisse (über 10.000 TP) nur erzielt werden, weil die zu bewertenden Elemente schon genau so in den Trainingsdaten gesehen wurden.

Die LSTM-Netze konnten R9 „Achtelnoten nur auf *Weak Beats*“:

$$P(N|Value(N) \wedge Pos(N) \wedge (S_t \wedge S_{t+1}))$$

37-mal erkennen. R9 wird in den Daten 128-mal gebrochen. Es kann vermutet werden, dass die Netze nur die Noten erkennen konnten, die auf den **DBeat** (also die Notenposition, die direkt in der Achtelnotenscheibe gekennzeichnet ist) fallen. Für das ZL-Netz ist anzunehmen, dass es erkennt, dass die mittlere Note in der Sequenz die betrachtete Note ist.

	DL	ZL	FF	DL	ZL	FF
	TP			FP		
R1	733	722	754	9	14	15
R2	10862	10876	10871	61	20	32
R3	7125	7155	7155	24	40	27
R4	635	643	639	24	28	11
R5	875	899	944	88	135	67
R6	2138	2211	2244	25	95	24
R7	0	0	0	0	0	0
R8	30	29	45	4	2	3
R9	29	37	91	2	5	11
	FN			TN		
R1	153	164	132	46807	46802	46801
R2	27	13	18	36752	36793	36781
R3	53	23	23	40500	40484	40497
R4	54	46	50	46989	46985	47002
R5	213	189	144	46526	46479	46547
R6	126	53	20	45413	45343	45414
R7	0	0	0	47702	47702	47702
R8	22	23	7	47646	47648	47647
R9	23	91	37	47648	47569	47563

Abb. 55 Confusion Matrix: postseparierte Daten.

Farbverlauf jeweils pro Regel und Viertel der Matrix (3 Werte)

*Für Regel R7 ist kein Farbverlauf eingetragen, da diese Regel in den postseparierten Daten nicht verletzt wird.

	DL	ZL	FF
i	40	79	680
h	312	312	312
o	1	1	1
hl	2	2	2
b	50	50	50
ep	30	30	30
do	0.3	0.3	0.0
lr	0.0001	0.0001	0.0001

Abb. 56 Einstellungen der Netze.

i: Input-Neuronen, h: Hidden-Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainiert.

Anhand dieser Versuchsreihe kann kein Urteil gefällt werden, ob die Netze in der Lage sind, Strukturen in den Kontrapunkten zu erkennen.

A3 Versuchsreihe „Training mit präseparierten Mutationsdaten“

Versuchsbeschreibung:

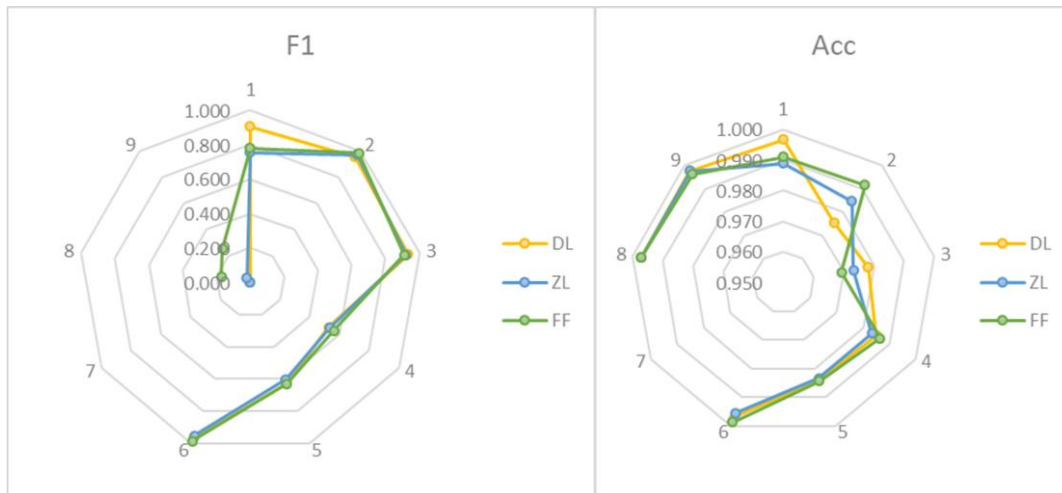
Die DL-, ZL- und FF-Netze sollen mit den präseparierten Mutationsdaten trainiert werden, um herauszufinden, ob der gute Lernerfolg mit den postseparierten Mutationstaten nur durch das Erkennen der in den Trainingsdaten vorhandenen Elemente erreicht wurde. Dabei soll jeweils pro Regel ein Netz trainiert werden, das die jeweilige Regel erkennen soll.

Trainingsdaten: präseparierte Mutationstrainingsdaten

Testdaten: präseparierte Mutationstestdaten

Validationsdaten: präseparierte Mutationsvalidationsdaten

Ergebnisse:



Wird mit den postseparierten Daten trainiert, brechen die Ergebnisse für die Regeln R8 und R9 komplett ein und auch R4 und R5 können nur noch mittlere Ergebnisse erzielen.

Beachtlich ist, dass das DL-Netz für R1 „Start- und Endregel“:

$$P(I_t | N_{t-1} = \emptyset \vee N_{t+1} = \emptyset)$$

sogar einen geringfügig besseren F1-Score erreichen konnte als in der Versuchsreihe mit den postseparierten Daten.

	F1			Acc		
	DL	ZL	FF	DL	ZL	FF
R1	0.907	0.754	0.782	0.997	0.989	0.991
R2	0.953	0.971	0.983	0.976	0.985	0.991
R3	0.935	0.922	0.911	0.978	0.973	0.969
R4	0.527	0.535	0.562	0.985	0.983	0.986
R5	0.626	0.603	0.628	0.984	0.984	0.984
R6	0.967	0.952	0.986	0.997	0.995	0.999
R7*						
R8	0.000	0.000	0.172	0.997	0.997	0.997
R9	0.000	0.032	0.253	0.998	0.997	0.996
Σ	4.915	4.769	5.277	7.911	7.904	7.915

Abb. 57 Farbverlauf jeweils für alle F1-Scores (24 Werte), Acc (24 Werte) und die jeweiligen Summen (3 Werte).

Die Summe soll nur einen Anhaltspunkt geben, wie die Netze im Vergleich zueinander abgeschnitten haben, sie eignet sich nicht zur qualitativen Bewertung.

*Für Regel R7 sind kein F1-Score und keine Acc angegeben, da diese Regel in den präseparierten Daten nicht verletzt wird.

Die in Kapitel 6.2 „Training mit prä- und postseparierten Mutationsdaten“ S. 34 besprochenen Ergebnisse zu R3 lassen vermuten, dass die Netzarchitekturen geeignet sind, die Kontrapunktregeln zu erlernen.

Die nächste Versuchsreihe „Training mit stochastischen Daten“ soll Aufschluss darüber geben, ob die schlechten Ergebnisse von R8 und R9 nur von ungünstigen Trainingsdaten abhängen.

	DL	ZL	FF	DL	ZL	FF
	TP			FP		
R1	746	792	749	24	433	291
R2	11398	11842	11751	563	588	179
R3	7317	7357	7356	937	1208	1394
R4	389	444	404	469	598	415
R5	626	585	612	372	352	336
R6	2102	2085	2154	45	96	17
R7	0	0	0	0	0	0
R8	0	0	14	0	0	12
R9	0	2	30	0	9	94
	FN			TN		
R1	129	83	126	45736	45327	45469
R2	572	128	219	34102	34077	34486
R3	82	42	43	38299	38028	37842
R4	230	175	215	45547	45418	45601
R5	376	417	390	45261	45281	45297
R6	97	114	45	44391	44340	44419
R7	0	0	0	46635	46635	46635
R8	137	137	123	46498	46498	46486
R9	113	111	83	46522	46513	46428

Abb. 58 Confusion Matrix: postseparierte Daten. Farbverlauf jeweils pro Regel und Viertel der Matrix (3 Werte)

*Für Regel R7 ist kein Farbverlauf eingetragen, da diese Regel in den präseparierten Daten nicht verletzt wird.

	DL	ZL	FF
i	40	79	680
h	312	312	312
o	1	1	1
hl	2	2	2
b	50	50	50
ep	30	30	30
do	0.3	0.3	0.0
lr	0.0001	0.0001	0.0001

Abb. 59 Einstellungen der Netze.

i: Input-Neuronen, h: Hidden-Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainiert. (Einstellungen der Versuchsreihen A2-A4 sind identisch)

A4 Versuchsreihe „Training mit stochastischen Daten“

Versuchsbeschreibung:

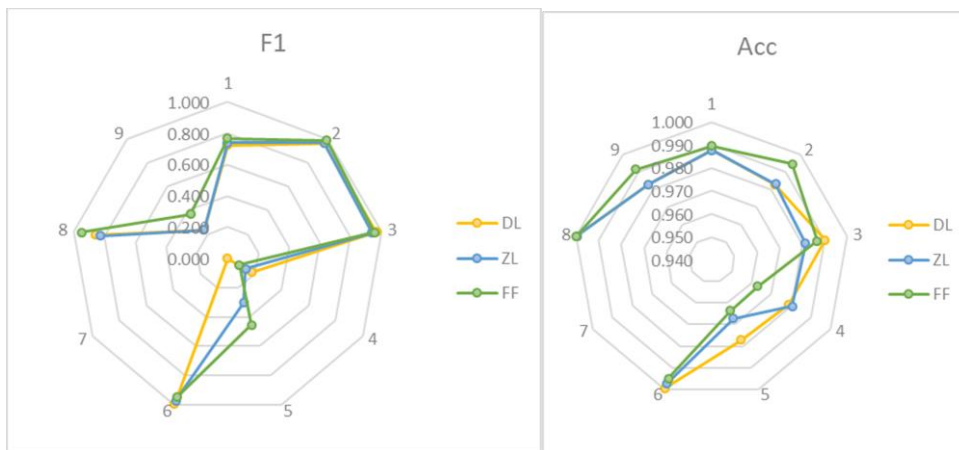
Die DL-, ZL- und FF-Netze sollen mit den stochastischen Daten trainiert werden, um herauszufinden, ob die Kontrapunktregeln von den Netzen gelernt werden können. Es soll auf den stochastischen Daten trainiert werden und anschließend auf den postseparierten Mutationsdaten geprüft werden, ob die Regeln auch in Kontrapunkten erkannt werden, die Ähnlichkeit mit realen Kontrapunkten haben. Dabei soll jeweils pro Regel ein Netz trainiert werden, das die jeweilige Regel erkennen soll.

Trainingsdaten: stochastische Trainingsdaten

Testdaten: postseparierte Mutationstestdaten

Validationsdaten: postseparierte Mutationsvalidationsdaten

Ergebnisse:



Wird mit den stochastischen Daten trainiert, können für R4 und R5 keine guten Ergebnisse mehr erzielt werden.

Die auch schon in den Versuchen mit den postseparierten Daten nur mittelmäßig erlernbare Regel R9 „Achtelnoten nur auf *Weak Beats*“:

$$P(N|Value(N) \wedge Pos(N) \wedge (S_t \wedge S_{t+1}))$$

kann auch mit den 37.521 Negativbeispielen der stochastischen Daten nicht zuverlässig erlernt werden. Diese Regel benötigt das Sprungintervall von der letzten zur betrachteten sowie das von der

	F1			Acc		
	DL	ZL	FF	DL	ZL	FF
R1	0.723	0.740	0.768	0.988	0.988	0.990
R2	0.964	0.964	0.988	0.983	0.983	0.995
R3	0.968	0.941	0.957	0.990	0.981	0.987
R4	0.178	0.133	0.087	0.979	0.981	0.963
R5	0.002	0.304	0.454	0.977	0.967	0.964
R6	0.994	0.970	0.945	0.999	0.997	0.995
R7*						
R8	0.863	0.831	0.950	1.000	1.000	1.000
R9	0.237	0.237	0.370	0.983	0.983	0.991
Σ	4.929	5.122	5.519	7.899	7.880	7.884

Abb. 60 Farbverlauf jeweils für alle F1-Scores (24 Werte), Acc (24 Werte) und die jeweiligen Summen (3 Werte). Die Summe soll nur einen Anhaltspunkt geben, wie die Netze im Vergleich zueinander abgeschnitten haben, sie eignet sich nicht zur qualitativen Bewertung.

*Für Regel R7 sind kein F1-Score und keine Acc angegeben, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.

betrachteten Note ausgehende Sprungintervall zur nächsten Note – vielleicht ist diese Regel schwer zu erlernen.

Wie schon in der Versuchsreihe „Training mit postseparierten Mutationsdaten“ beschrieben, gehen die 128 als TP erkannten Elemente vermutlich auf das Erkennen des **DBeats** zurück, das würde auch die hohe Anzahl (822) an FPs erklären.

Die hohen F1-Scores für R2, R3 und R6, in Zusammenhang mit der vergleichsweise geringen Anzahl der $|d_b^{Rx}|$ für 5 Achtelnotenscheiben (R2: 2176, R3: 1667, R6: 515) lassen vermuten, dass für diese Regeln tatsächlich gelernt wurde, die nötigen Konzepte auszuwerten.

Durch die Teilregel von R1: „Wenn der **c.p.** im Bass steht, muss er mit derselben Note beginnen, mit der der **c.f.** endet“ können die Netze mit ihrem Input nicht alle Elemente eindeutig bewerten, was diese Regel angeht (denn sie „sehen“ die letzte **c.f.**-Note nicht).

R8 erreicht zwar einen hohen F1-Score, liegt aber, werden die stochastischen Trainingsdaten mit den postseparierten Validationsdaten verglichen, im 5-Achtelnotenbereich 35-mal als Duplikat vor.

	DL	ZL	FF	DL	ZL	FF
	TP			FP		
R1	755	830	813	448	526	419
R2	10875	10840	10841	808	750	211
R3	7130	7120	7098	422	830	557
R4	109	71	84	430	310	1152
R5	1	342	720	3	819	1366
R6	2246	2149	2056	7	19	32
R7	0	0	0	0	0	0
R8	41	37	48	2	0	1
R9	128	128	119	822	822	397
	FN			TN		
R1	131	56	73	46368	46290	46397
R2	14	49	48	36005	36063	36602
R3	48	58	80	40102	39694	39967
R4	580	618	605	46583	46703	45861
R5	1087	746	368	46611	45795	45248
R6	18	115	208	45431	45419	45406
R7	0	0	0	47702	47702	47702
R8	11	15	4	47648	47650	47649
R9	0	0	9	46752	46752	47177

Abb. 61 Confusion Matrix: postseparierte Daten. Farbverlauf jeweils pro Regel und Viertel der Matrix (3 Werte)

*Für Regel R7 ist kein Farbverlauf eingetragen, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.

	DL	ZL	FF
i	40	79	680
h	312	312	312
o	1	1	1
hl	2	2	2
b	50	50	50
ep	30	30	30
do	0.3	0.3	0.0
lr	0.0001	0.0001	0.0001

Abb. 62 Einstellungen der Netze.

i: Input-Neuronen, h: Hidden Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainiert. (Einstellungen der Versuchsreihen A2-A4 sind identisch)

Aufgrund des sehr schlechten Ergebnisses des LSTM DL beim Bewerten von R5 wurde der Test wiederholt. Das schlechte Ergebnis ließ sich dabei reproduzieren. (Abb. 63)

Im Anschluss wurde versucht, das noch nicht ganz ausgelernt zu sein scheinende ZL-Netz zu verbessern.

Da keins der untersuchten Netze eine signifikante Verbesserung erreichen konnte, wird hier nur kurz das Netz mit dem höchsten F1-Score (0,433) vorgestellt. Dieser liegt immer noch unter dem Höchstwert des FF (0,454).

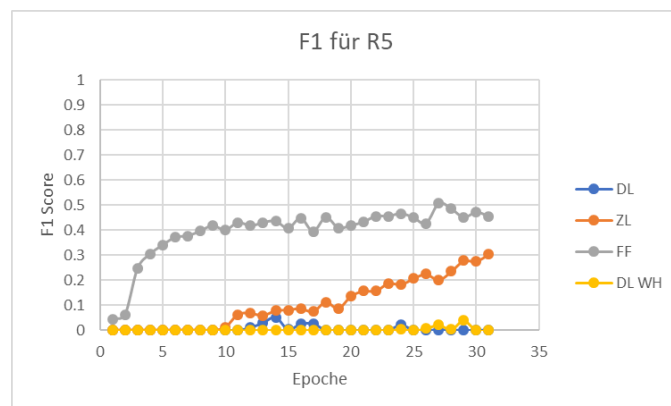


Abb. 63 Wiederholung (DL WH) des Trainings für Regel R5.

Die in Abb. 64 dargestellten Werte zeigen, dass das Netz sich ungefähr ab Epoche 70 nicht mehr stark verbessert.⁴⁹

In der folgenden Versuchsreihe soll getestet werden, ob die schlechten Ergebnisse von R5 mit einer anderen Lernmethode verbessert werden können.

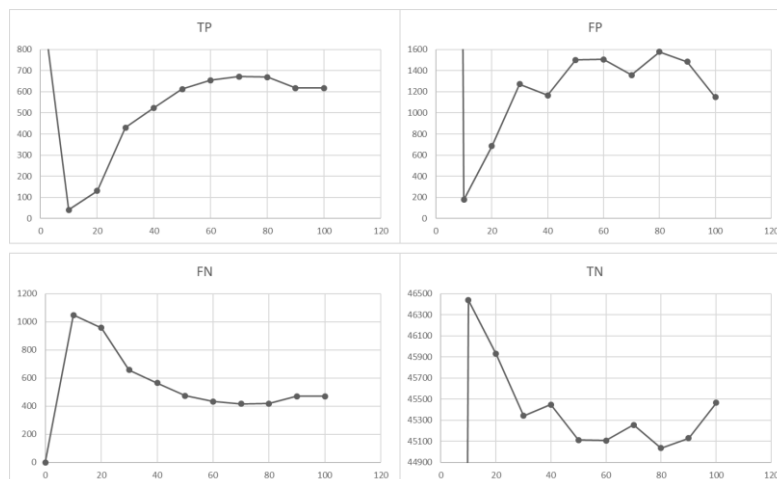


Abb. 64 Die Confusion Matrix dargestellt über den Verlauf von 100 Epochen.

⁴⁹ Wegen des erhöhten Werts der TNs in Epoche 100 wurde das Netz noch ein zweites Mal bis Epoche 300 trainiert, dies brachte aber keine Verbesserung.

A5 Versuchsreihe „Training einzelner Netze auf alle Regeln“

Versuchsbeschreibung:

Die DL-, ZL- und FF-Netze sollen mit den stochastischen Daten trainiert werden. Da die Regeln teils auf denselben Grundbausteinen der Musiktheorie beruhen, soll geprüft werden, ob das Lernen aller Regeln von Vorteil für den Lernerfolg der Netze ist.

Die Annahme hierbei ist, dass eine Regel, die in den Trainingsdaten nur selten verletzt wird, besser gelernt werden kann, wenn durch eine andere Regel schon die hierfür benötigten Muster gelernt wurden. So könnten die durch R2 „Bevorzugte *Motion*“ erlernten Muster für R5 „Bevorzugte *Motion*-abhängige *Intervals*“ eine Verbesserung bringen.

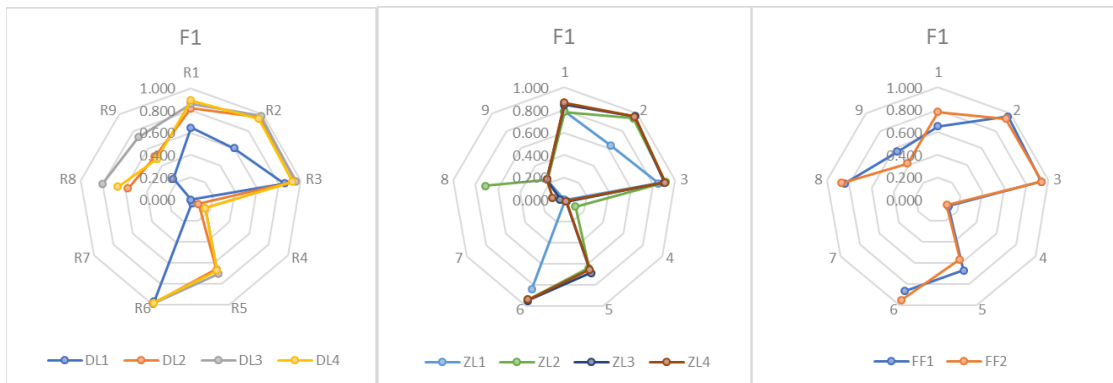
Andererseits könnte die erhöhte Komplexität des Problems auch das Gegenteil erreichen und gar kein Lernerfolg mehr stattfinden.

Trainingsdaten: stochastische Trainingsdaten

Testdaten: postseparierte Mutationstestdaten

Validationsdaten: postseparierte Mutationsvalidationsdaten

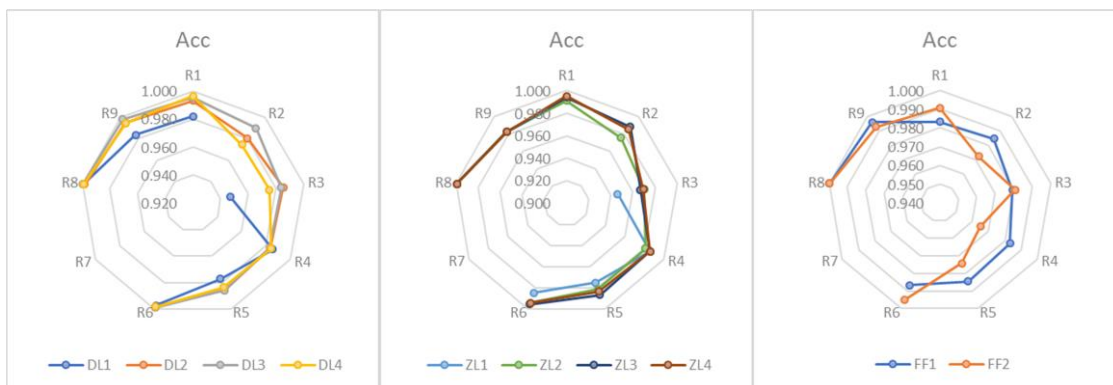
Ergebnisse:



	F1									
	LSTM durchlaufend				LSTM zulaufend				Feedforward	
	DL1	DL2	DL3	DL4	ZL1	ZL2	ZL3	ZL4	FF1	FF2
R1	0.646	0.817	0.861	0.890	0.789	0.784	0.846	0.863	0.654	0.780
R2	0.602	0.958	0.978	0.945	0.636	0.949	0.975	0.969	0.967	0.942
R3	0.850	0.953	0.949	0.923	0.847	0.910	0.900	0.907	0.935	0.938
R4	0.000	0.077	0.147	0.149	0.000	0.113	0.023	0.017	0.109	0.089
R5	0.032	0.664	0.708	0.675	0.013	0.641	0.691	0.659	0.668	0.565
R6	0.970	0.986	0.989	0.982	0.847	0.941	0.953	0.941	0.864	0.948
R7**										
R8	0.000	0.573	0.806	0.662	0.000	0.713	0.038	0.109	0.839	0.871
R9	0.244	0.504	0.730	0.476	0.237	0.237	0.237	0.237	0.560	0.420
Σ	3.344	5.532	6.169	5.703	3.369	5.287	4.663	4.703	5.596	5.553

Abb. 65 Farbverlauf jeweils für alle F1-Scores (80 Werte) und die jeweiligen Summen (10 Werte).

**Für Regel R7 ist kein F1-Score und angegeben, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.



	ACC									
	LSTM durchlaufend				LSTM zulaufend				Feedforward	
	DL1	DL2	DL3	DL4	ZL1	ZL2	ZL3	ZL4	FF1	FF2
R1	0.982	0.993	0.995	0.996	0.992	0.991	0.994	0.995	0.983	0.991
R2	0.822*	0.980	0.990	0.974	0.803*	0.976	0.988	0.986	0.985	0.972
R3	0.947	0.985	0.984	0.975	0.946	0.970	0.967	0.969	0.979	0.981
R4	0.986	0.983	0.983	0.984	0.986	0.981	0.986	0.986	0.983	0.965
R5	0.977	0.986	0.986	0.984	0.975	0.982	0.987	0.984	0.985	0.974
R6	0.997	0.999	0.999	0.998	0.985	0.995	0.996	0.995	0.987	0.995
R7**										
R8	0.999	0.998	0.999	0.999	0.999	0.999	0.999	0.999	1.000	1.000
R9	0.983	0.995	0.998	0.994	0.983	0.983	0.983	0.983	0.996	0.993
Σ	7.693	7.920	7.934	7.905	7.668	7.876	7.899	7.896	7.898	7.871

Abb. 66 Farbverlauf jeweils für alle Acc (80 Werte) und die jeweiligen Summen (10 Werte).

*Diese Werte sind in den Netzdiagrammen nicht abgebildet.

**Für Regel R7 ist keine Acc angegeben, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.

	DL1	DL2	DL3	DL4	ZL1	ZL2	ZL3	ZL4	FF1	FF2
i	40	40	40	40	79	79	79	79	680	680
h	312	312	624	312	312	312	624	312	312	312
o	9	9	9	9	9	9	9	9	9	9
hl	2	2	2	2	2	2	2	2	2	2
b	50	50	50	50	50	50	50	50	50	50
ep	100	100	100	100	100	100	100	100	100	100
do	0.3	0.0	0.3	0.3	0.3	0.0	0.3	0.3	0.0	0.0
lr	0.001	0.0001	0.0001	0.0001	0.001	0.0001	0.0001	0.0001	0.001	0.0001

Abb. 67 Einstellungen der Netze.

DL4, ZL4 und FF2 können als die Standardeinstellungen angesehen werden, die fettgedruckten Werte geben jeweils die Abweichung zu diesen Einstellungen an.

i: Input-Neuronen, h: hidden Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainiert.

Die LSTM-Netze, die in der Gesamtsumme des F1-Scores am besten abschneiden, werden hier weiter diskutiert (DL3, ZL2). Für die FF-Netze werden sowohl FF1, als auch FF2 diskutiert, da beide bei einzelnen Regeln höhere F1-Scores erzielten. Die schlechten F1-Scores von DL1 und ZL1 sind auf die kleinere *learning rate* zurückzuführen, diese beiden Netze werden hier nicht weiter besprochen. Da die Netze DL2, DL4, ZL3 und ZL4 in keiner Regel einen auffällig höheren F1-Score erzielen konnten als die Netze DL3 und ZL3, werden sie hier ebenfalls nicht weiter besprochen.

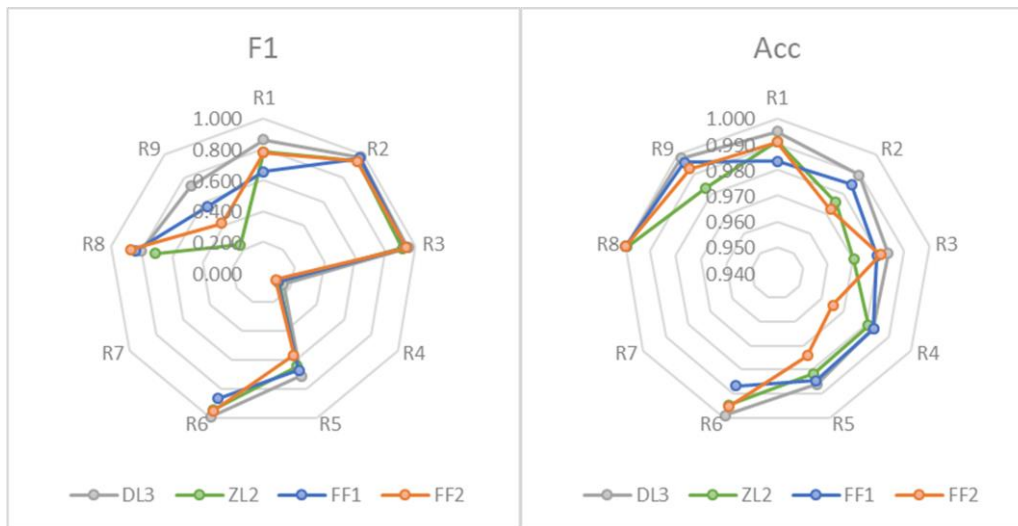


Abb. 68 Die in den Versuchen bestabschneidenden Netze.

	DL3	ZL2	FF1	FF2	DL3	ZL2	FF1	FF2
	TP				FP			
R1	752	780	756	782	108	324	669	338
R2	10811	10827	10601	10787	418	1103	440	1221
R3	7149	7164	7049	6997	743	1411	851	743
R4	69	58	49	81	180	277	157	1059
R5	822	781	742	799	411	569	393	939
R6	2223	2048	2020	2090	8	42	390	55
R7	0	0	0	0	0	0	0	1
R8	50	46	39	44	22	31	2	5
R9	119	126	121	123	79	811	183	335
	FN				TN			
R1	134	106	130	104	46708	46492	46147	46478
R2	78	62	288	102	36395	35710	36373	35592
R3	29	14	129	181	39781	39113	39673	39781
R4	620	631	640	608	46833	46736	46856	45954
R5	266	307	346	289	46203	46045	46221	45675
R6	41	216	244	174	45430	45396	45048	45383
R7	0	0	0	0	47702	47702	47702	47701
R8	2	6	13	8	47628	47619	47648	47645
R9	9	2	7	5	47495	46763	47391	47239

Abb. 69 Confusion Matrix: postseparierte Daten. Farbverlauf jeweils pro Regel und Viertel der Matrix (4 Werte)

*Für Regel R7 ist kein Farbverlauf eingetragen, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.

Die Ergebnisse zeigen, dass mit denselben Trainingsdaten, mit denen auch bei der Versuchsreihe „Training mit stochastischen Daten“ trainiert wurde, höhere F1-Scores für R1, R5 und R9 erreicht werden konnten.

Der Höchstwert des F1-Scores für R5 konnte von 0,454 (FF aus Versuchsreihe „Training mit stochastischen Daten“) auf 0,708 (DL3) um 0,254 verbessert werden.

Im Fall von R8 konnte als höchster F1-Score noch 0,806 (DL3) erreicht werden, dem gegenüber steht das FF der Versuchsreihe „Training mit stochastischen Daten“ mit 0,95. Eine Verschlechterung um 0,144.

In der nächsten Versuchsreihe soll geprüft werden, ob auch mit wenigen Trainingsdaten ein Erfolg erzielt werden kann.

A6 Versuchsreihe „Reduzieren der Trainingsbeispiele“

Versuchsbeschreibung:

Die DL-, ZL- und FF-Netze sollen mit ausgewählten stochastischen Daten trainiert werden. Dabei werden 1.000 Kontrapunkte ausgewählt, die die Regel beinhalten, gegen die am seltensten verstoßen wird. Anschließend soll iterativ die Regel in diesen selektierten Daten ausgewählt werden, gegen die dort am seltensten verstoßen wird. Dieser Vorgang soll solange stattfinden, bis in den selektierten Daten gegen jede der neun Regeln mindestens 1.000-mal verstoßen wird.

Durch das Reduzieren der Trainingsdaten soll geprüft werden, ob ein Lernerfolg auch mit wenigen Daten möglich ist und so die Praktikabilität für den tatsächlichen Einsatz in der Lehre geprüft werden.

Trainingsdaten: selektierte stochastische Trainingsdaten

Testdaten: postseparierte Mutationstestdaten

Validationsdaten: postseparierte Mutationsvalidationsdaten

Erstellen der Trainingsdaten:

Regel R8 ist diejenige, gegen die in den stochastischen Trainingsdaten am seltensten (1.141) verstoßen wird. Werden aus diesen Daten 1.000 ausgewählt, werden alle Regeln bis auf Regel R4 und R5 bereits 1.000-mal verletzt. Gegen Regel R8 wird an diesem Punkt 1.091-mal verstoßen, da in einigen Kontrapunkten mehrmals gegen R8 verstoßen wird. (Abb. 70)

Werden aus den übrigen Trainingsdaten nun Kontrapunkte ausgewählt, bis gegen R5 in 1.000 Kontrapunkten verstoßen wird, wird bereits nach der ersten Iteration gegen alle Regeln 1.000-mal verstoßen. (Abb. 71)

Insgesamt bestehen die selektierten Daten aus 1.233 Kontrapunkten.

Regel	Gebrochen
R1	1592
R2	9422
R3	7676
R4	945
R5	733
R6	8349
R7	5053
R8	1091
R9	3679

Abb. 70 Gebrochene Regeln nach Auswahl von R8.

Regel	Gebrochen
R1	1926
R2	11592
R3	9463
R4	1167
R5	1055
R6	10250
R7	6082
R8	1091
R9	4548

Abb. 71 Gebrochene Regeln nach der 1. Iteration.

Die selektierten Daten wurden ebenfalls auf die mehrfach vorhandenen Elemente überprüft (Abb. 72).

Ergebnisse:

d_b^{Rx} für Elemente aus 17 Achtelnotenscheiben										
a	b	R1	R2	R3	R4	R5	R6	R7	R8	R9
selected	post valid	1	0	0	0	0	0	0	0	0
d_b^{Rx} für Elemente aus 9 Achtelnotenscheiben										
selected	post valid	115	100	32	0	0	4	0	2	0
d_b^{Rx} für Elemente aus 5 Achtelnotenscheiben										
selected	post valid	186	655	390	1	5	75	0	35	0

Abb. 72 Die Elemente der selektierten Daten, die auch in den postseparierten Validationsdaten auftauchen, und wie oft diese Elemente in diesen postseparierten Validationsdaten vorhanden sind.



	F1						Acc					
	Alle Regeln			Einzelregeln			Alle Regeln			Einzelregeln		
	DLar	ZLar	FFar	DLsr	ZLsr	FFsr	DLar	ZLar	FFar	DLsr	ZLsr	FFsr
R1	0.358	0.330	0.433	0.354	0.355	0.460	0.938	0.945	0.964	0.938	0.940	0.963
R2	0.558	0.645	0.659	0.689	0.737	0.805	0.837	0.802	0.803	0.824	0.862	0.900
R3	0.513	0.498	0.798	0.798	0.850	0.876	0.727	0.722	0.928	0.928	0.950	0.959
R4	0.132	0.237	0.230	0.143	0.110	0.125	0.983	0.979	0.981	0.973	0.950	0.954
R5	0.000	0.000	0.169	0.000	0.037	0.271	0.977	0.977	0.967	0.977	0.976	0.957
R6	0.691	0.653	0.725	0.870	0.800	0.824	0.974	0.968	0.978	0.988	0.982	0.984
R7*												
R8	0.000	0.000	0.278	0.264	0.694	0.662	0.999	0.999	0.997	0.998	0.999	0.999
R9	0.214	0.237	0.237	0.237	0.236	0.239	0.980	0.983	0.983	0.983	0.983	0.983
Σ	2.465	2.601	3.529	3.356	3.819	4.262	7.415	7.375	7.599	7.610	7.642	7.700

Abb. 73 Farbverlauf jeweils für alle F1-Scores (48 Werte) bzw. die Acc (48 Werte) und die jeweiligen Summen (6 Werte).

*Für Regel R7 ist kein F1-Score und keine Acc angegeben, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.

Alle Netze wurden mit den Einstellungen aus Abb. 74 trainiert.

Die mit den wenigen selektierten Trainingsdaten trainierten Netze konnten die hohen F1-Scores der Netze, die mit den kompletten selektierten Daten trainiert wurden, nicht erreichen.

Die Ergebnisse für R2, R3 und R6 sind jedoch noch annehmbar. Für diese drei Regeln liegen in den selektierten Daten allerdings auch die meisten Negativbeispiele vor (≈ 10.000 pro Regel).

Zu beachten ist, dass für R4 in dieser Versuchsreihe der höchste F1-Score beim Training mit stochastischen

Trainingsdaten erreicht werden konnte. Das alle Regeln lernende ZL-Netz konnte einen Score von 0,237 erzielen. Der nächsthöchste Wert 0,178 (abgesehen von FF AR dieser Versuchsreihe) wurde in der Versuchsreihe „Training mit stochastischen Daten“ vom DL-Netz erreicht. Die Differenz beträgt 0,059: eine Steigerung von 24,89%.

	DLar	ZLar	FFar	DLsr	ZLsr	FFsr
i	40	79	680	40	79	680
h	624	312	312	312	312	312
o	9	9	9	1	1	1
hl	2	2	2	2	2	2
b	50	50	50	50	50	50
ep*	30	30	30	30	30	30
do	0.3	0.0	0.0	0.3	0.3	0.0
lr	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001

Abb. 74 Einstellungen der Netze.

*Die Versuche wurden mit 100 Epochen wiederholt, es konnten aber keine besseren Ergebnisse erzielt werden. Für die 100 Epochen-Netze wurden keine Zwischenergebnisse gesammelt, deshalb werden hier nur die 30 Epochen-Netze besprochen.

i: Input-Neuronen, h: hidden Neuronen pro Layer, o: Output-Neuronen, hl: Hidden-Layer-Anzahl, b: batch size, ep: Epochen, do: dropout, lr: learning rate. Alle Netze wurden mit einem ADAM-Optimizer und dem Binary Cross Entropy Loss trainierte.

	Alle Regeln			Einzelregeln				Alle Regeln			Einzelregeln		
	DLar	ZLar	FFar	DLsr	ZLsr	FFsr		DLar	ZLar	FFar	DLsr	ZLsr	FFsr
	TP							FP					
R1	822	645	664	809	788	752		2884	2373	1517	2872	2763	1633
R2	4905	8548	9058	9286	9241	9836		1778	7088	7554	6788	4934	3722
R3	6860	6583	6802	6776	6791	6950		12713	12655	3057	3027	2007	1731
R4	63	156	139	106	147	156		201	469	379	685	1827	1658
R5	0	0	161	0	22	382		0	0	660	0	94	1344
R6	1377	1454	1410	1910	1675	1754		347	737	214	217	251	240
R7	0	0	0	0	0	0		0	0	1	0	0	0
R8	0	0	30	14	43	45		0	0	134	40	29	39
R9	128	128	127	128	127	124		943	822	817	822	822	785
	FN							TN					
R1	64	241	222	77	98	134		43932	44443	45299	43944	44053	45183
R2	5984	2341	1831	1603	1648	1053		35035	29725	29259	30025	31879	33091
R3	318	595	376	402	387	228		27811	27869	37467	37497	38517	38793
R4	626	533	550	583	542	533		46812	46544	46634	46328	45186	45355
R5	1088	1088	927	1088	1066	706		46614	46614	45954	46614	46520	45270
R6	887	810	854	354	589	510		45091	44701	45224	45221	45187	45198
R7	0	0	0	0	0	0		47702	47702	47701	47702	47702	47702
R8	52	52	22	38	9	7		47650	47650	47516	47610	47621	47611
R9	0	0	1	0	1	4		46631	46752	46757	46752	46752	46789

Abb. 75 Confusion Matrix: postseparierte Daten. Farbverlauf jeweils pro Regel und Viertel der Matrix (6 Werte).

*Für Regel R7 ist kein Farbverlauf eingetragen, da diese Regel in den postseparierten Validationsdaten nicht verletzt wird.

B Ausschnitt 1 aus der Dokumentation der vorangegangenen Arbeit

Dieser Ausschnitt wurde im Rahmen des vorangegangenen Praxisprojektes [Scholzen, 2019] erstellt und wird hier zum besseren Verständnis für den Leser noch einmal wiedergegeben.

B1 Bewertung der Noten

Bei den formulierten Regeln gibt es solche, die mögliche Folgenoten komplett ausschließen (harte Regeln), aber auch solche, die mögliche Folgenoten nur unwahrscheinlicher machen (weiche Regeln).

Um die Regeln formulieren zu können, wurden Wahrscheinlichkeitstabellen entworfen, die den Regeln ihre Wichtigkeit zuordnen. Diese Wahrscheinlichkeiten sind rein subjektiv und dem Wortlaut des Buches nachempfunden.

Aus Sätzen wie “[...] the beginning and the end [...] must consist of perfect consonances.” wird, sollte die Regel gebrochen sein, eine nullprozentige Wahrscheinlichkeit abgeleitet. Sätze wie “[...] if the two parts have been let so close together that one does not know where to take them; and if there is no possibility of using contrary motion, this motion can be brought about by using the skip of a minor sixth [...] or an octave” führen dazu, dass die Grundwahrscheinlichkeit niedrig ist, sie aber unter erfüllten Voraussetzungen steigt.

Die gewählte Notation ist der von bedingten Wahrscheinlichkeiten, wie sie in [Russell & Norvig, 2012, S. S. 574 ff.] verwendet werden, nachempfunden.

Alle Werte, die nicht in den Tabellen aufgeführt sind, ergeben sich zu $P(x) = 0.0$, wobei x die betrachtete Eigenschaft einer Note ist (zum Beispiel das *Interval* zum *c.f.*). Eine Wahrscheinlichkeit von $P(x) = 0.0$ bedeutet, dass die Note ausgeschlossen werden kann, da sie den harten Regeln des Kontrapunkts widerspricht. Jeder Wert $0 < P(x) < 1.0$ bedeutet, dass die Note eine weiche Regel verletzt und deshalb unwahrscheinlicher wird. Jeder Wert $P(x) = 1.0$ bedeutet folglich, dass die Note allen Regeln des Kontrapunkts entspricht.

Müssen Bedingungen im Verhältnis zu Vornote beschrieben werden, wird dies durch die Notation $P(x_t|x_{t-1})$ ausgedrückt. Hierbei ist x_t die aktuell betrachtete Eigenschaft einer Note (z.B. in welchem *Interval* sie zum *c.f.* steht), x_{t-1} die Eigenschaft der letzten Note.

Will man die Wahrscheinlichkeit des *Intervals* der betrachteten Note in Abhängigkeit des *Intervals* der vorherigen Note beschreiben, würde gelten:

$$P(I_t|I_{t-1}) = p \Rightarrow P(P8|m6) = 1.0$$

$$I := \text{Interval}, x = I \text{ und } I_t = P8, I_{t-1} = m6$$

Aussage: Die Wahrscheinlichkeit eines P8 Intervals beträgt 100%, falls das letzte Interval eine m6 war.

Werden unterschiedliche Eigenschaften betrachtet, wird dies durch $P(x_t|y_{t-1})$ ausgedrückt (z.B. $M := \text{Motion}; x = M, y = I; M_t = \text{CoMo}, I_{t-1} = M7$).

Kommen für eine Eigenschaft mehrere Werte infrage, wird dies durch aussagenlogische Operatoren dargestellt.

$$P(x_t|y_t), \quad P(M_t|I_t), \quad P(\text{CoMo}|I_t = 5 \vee 8) = 1.0$$

Aussage: Die Wahrscheinlichkeit, dass CoMo stattfinden kann, wenn gleichzeitig das Intervall P5 oder P8 erzeugt wird, beträgt 100%.

Muss zwischen Eigenschaften des **c.f.** und des **c.p.** unterschieden werden, wird dies durch die Subskripte c.p. und c.f. notiert ($x_{c.p.}, y_{c.f.}$).

B1.1 Annahmen über die Wahrscheinlichkeiten der Noten

Es folgen die getroffenen Annahmen über die Wahrscheinlichkeitswerte der einzelnen Regeln. Werden mehrere Regeln verletzt, wird im Programm das Produkt der Wahrscheinlichkeiten gebildet, um die Gesamtwahrscheinlichkeit zu berechnen. Werden in den Tabellen Intervalle und Notensprünge nicht aufgelistet, haben sie immer die Wahrscheinlichkeit 0.0.

B1.2 R1 Start- und Endregel

R1 Start und Ende müssen *perfect consonances* sein. Wenn der CP im Bass steht, muss er mit derselben Note beginnen, mit der der *c.f.* endet.

Mögliche <i>Intervals</i>	Erzeugtes <i>Interval</i>	Wahrscheinlichkeit
$P(I_t N_{t-1} = \emptyset \vee N_{t+1} = \emptyset)$	8	1.0
	6	0.0
	5	0.5
	4	0.0
	3	0.0
	1	1.0

Abb. 76 Die Wahrscheinlichkeit für das Intervall der betrachteten Note unter der Bedingung, dass es keine Vorgängernote gibt.

B1.3 R2 Bevorzugte Motion

R2 CoMo sollte bevorzugt werden (geht aus Fux' Beispielen hervor), **ObMo** ist auch in Ordnung, **DiMo** sollte vermieden werden.

R2 ist für $N_{t-1} = \emptyset$ nicht definiert, da hier keine Motion stattfindet.

Mögliche <i>Motions</i>	Erzeugte <i>Motion</i>	Wahrscheinlichkeit
$P(M Pos(N) = DBeat)$	CoMo	1.0
	DiMo	0.2
	ObMo	0.9

Abb. 77 Es sollte so oft wie möglich versucht werden, CoMo und ObMo zu erreichen.

B1.4 R3 Bevorzugte Intervalle

R3 *imperfect consonances* sollten bevorzugt werden. Keine *dissonants* auf dem *down beat*.

R3a *Dissonants* auf dem *Down Beat* sind erlaubt, wenn es *Tied Notes* sind und sie richtig, d.h. mit einem *Step down* in ein *constant interval*, aufgelöst werden.

Mögliche Intervals	Erzeugtes Interval	Wahrscheinlichkeit
$P(I N_{t-1} \neq \emptyset \wedge N_{t+1} \neq \emptyset)$	8	0.7
	6	1.0
	5	0.7
	4	0.0
	3	1.0
	1	0.7

Abb. 78 Unter der Voraussetzung, dass die Bedingung von R1 nicht zutrifft, also die Note nicht die erste oder letzte Note ist, sollten dissonant intervals bevorzugt werden.

Mögliche Intervals	S_{t+1}	I_{t+i}	Erzeugtes Interval I_t	Wahrscheinlichkeit
$P(I_t \text{Tied}(N_t) = \text{true} \wedge \text{Pos}(N_t) = \text{DBeat} \wedge I_{t+1} \wedge S_{t+1} \wedge N_{t-1} \neq \emptyset \wedge N_{t+1} \neq \emptyset)$	$s_{t+1} = 2$	$i_{t+1} = CI$	$\forall i \in I$	1.0
		$i_{t+1} \neq CI$	CI	1.0
			DisI	0.0
	$s_{t+1} > 2$	$i_{t+1} = CI$	CI	1.0
			DisI	0.0
		$i_{t+1} \neq CI$	$\forall i \in I$	0.0

Abb. 79 *Tied Notes* dürfen nur verwendet werden, wenn sie auf einem *DBeat* stehen. Sie dürfen außerdem dissonant sein, wenn sie richtig aufgelöst werden. Werden sie nicht richtig aufgelöst, scheiden die dissonanten Intervalle aus.
 S_t : Skip Interval

B1.5 R4 Einleiten der Schlussnote

R4 Die vorletzte Note muss eine M6 mit dem *c.f.* bilden, wenn der *c.f.* im Bass steht und eine m3, wenn er im Soprano steht.

R4a Die Note, die die Vorletzte einleitet, muss ein P5 sein, wenn der *c.f.* im Bass steht.

Mögliche <i>Intervals</i>	Erzeugtes <i>Interval</i>	Wahrscheinlichkeit
$P(I_t N_{t+2} = \emptyset \wedge \text{bass}(c.f.) = \text{true})$	8	0.0
	M6	0.0
	5	0.0
	4	0.0
	m3	1.0
	1	0.0
$P(I_t N_{t+2} = \emptyset \wedge \text{bass}(c.f.) = \text{false})$	8	0.0
	M6	1.0
	5	0.0
	4	0.0
	m3	0.0
	1	0.0

Abb. 80 Für die vorletzte Note gibt es nur je eine Möglichkeit je nachdem, ob der *c.f.* im Bass oder Soprano steht.

Mögliche <i>Intervals</i>	Erzeugtes <i>Interval</i>	Wahrscheinlichkeit
$P(I_t N_{t+3} = \emptyset \wedge \text{bass}(c.f.) = \text{true})$	8	0.0
	M6	0.0
	5	1.0
	4	0.0
	m3	0.0
	1	0.0

Abb. 81 Steht der *c.f.* im bass, ist das Intervall der vorletzten Note immer vorgegeben.

B1.6 R5 Bevorzugte Motion-abhängige Intervals

R5 Von einem *imperfect Interval* zu einem *imperfect Interval* ist jede *Motion* erlaubt, von einem *imperfect Interval* zu einem *perfect Interval* darf nur in **CoMo** geschritten werden. Von einem *perfect Interval* darf nur in **CoMo** zu einem weiteren *perfect Interval* geschritten werden. Von einem *perfect Interval* zu einem *imperfect* darf jede *Motion* verwendet werden. Aus den Beispielen im Buch geht hervor, dass dies nur die *Motions* zu den *Down Beats* betrifft.

Bevorzugte Intervals	I_{t-1}	M_t	I_t	Wahrscheinlichkeit
$P(I_t I_{t-1} \wedge M_t)$	PerCI	CoMo	PerCI	1.0
			iPerCI	1.0
		ObMo	PerCI	0.0
			iPerCI	1.0
		DiMo	PerCI	0.0
			iPerCI	1.0
	iPerCI	CoMo	PerCI	1.0
			iPerCI	1.0
		ObMo	PerCI	0.0
			iPerCI	1.0
		DiMo	PerCI	0.0
			iPerCI	1.0

Abb. 82 Es ergibt sich, dass zu einem *perfect Interval* nur in *Contrary Motion* geschritten werden darf und zu einem *imperfect Interval* in jeder der *Motions*. I_t ist für diese Regel also unabhängig von I_{t-1} .

Durch die (in der Tabelle ablesbare) Unabhängigkeit von I_{t-1} ergibt sich die vereinfachte Tabelle:

Bevorzugte Intervals	M_t	I_t	Wahrscheinlichkeit
$P(I_t M_t \wedge Pos(N_t) = DBeat)$	CoMo	PerCI	1.0
		iPerCI	1.0
	ObMo	PerCI	0.0
		iPerCI	1.0
	DiMo	PerCI	0.0
		iPerCI	1.0

Abb. 83 Vereinfachte Motion-abhängige Intervals.

B1.7 R6 Verbotene Skips

R6 Keine Triton *skips*. Keine M6 *skips*. Keine Notensprünge von einem P1 *Interval* in eine P5 oder P8.

R6a Große Sprünge ($s > \mathbf{M3}$) nur in eine m6 oder P8. Aus den Beispielen geht hervor, dass auch eine P5 erlaubt zu sein scheint.

Verbotene Notensprünge	I_{t-1}	S	Wahrscheinlichkeit
$P(S I_{t-1})$	$i \neq P1$	$s > \mathbf{M3}$ $\wedge s \neq P5$ $\wedge s \neq m6$ $\wedge s \neq P8$	0.0
		$s \leq \mathbf{M3}$ $\vee s = P5$ $\vee s = m6$ $\vee s = P8$	1.0
	$i = P1$	$s > \mathbf{M3}$ $\wedge s \neq m6$	0.0
		$s \leq \mathbf{M3}$ $\vee s = m6$	1.0

Abb. 84 Notensprünge größer als M3 sind grundsätzlich verboten (Tri und M6 sind in $s > M3$ enthalten).
Ausnahmen bildet die m6 und, falls nicht von der P1 gesprungen wird, die P5 und P8.

B1.8 R7 Ties nur auf dem Down Beat

R7 Mit einer *Tie* verbundene Noten dürfen nur auf dem *DBeat* stehen.

Tied Note möglich	$Pos(N)$	$Tied(N)$	Wahrscheinlichkeit
$P(Tied(N) Pos(N))$	DBeat	True	1.0
		False	1.0
	UBeat	True	0.0
		False	1.0
	WBeat	True	0.0
		False	1.0

Abb. 85 Tied Notes sind nur auf dem DowBeat erlaubt.

B1.9 R8 Verbotene Suspensions

R8 Ist eine Note *Tied*, sollte sie nicht von einer P1 zu einer m/M2 oder von einer m/M2 zu einer P8 springen, wenn der *c.f.* in der unteren Stimme steht. Steht der *c.f.* im Soprano, sollte nicht von einer m/M7 in eine P8 aufgelöst werden.

Mögliche <i>Intervals</i>	<i>Voice(c.f.)</i>	I_{t-1}	Erzeugtes <i>Interval I_t</i>	Wahrscheinlichkeit
$P(I_t Tied(N_t) = true$ $\wedge Voice(c.f.)$ $\wedge I_{t-1})$	Soprano	$i = m7$ $\vee i = M7$	P8	0.5
	Bass	$i = P1$	$i = m2$ $\vee i = M2$	0.5
		$i = m2$ $\vee i = M2$	P8	0.5

Abb. 86 Alle nicht aufgeführten Werte ergeben sich zu 1.0.

B1.10 R9 Achtelnoten nur auf Weak Beats

R9 *Eighths* sind nur auf den *WBeats* erlaubt. In sie sollte außerdem nur mit einem *Step* hinein und hinaus gesprungen werden.

Verbotene Achtelnoten	<i>Pos(N)</i>	$(S_t \wedge S_{t+1})$	Wahrscheinlichkeit
$P(N Value(N) = Eight$ $\wedge Pos(N)$ $\wedge (S_t \wedge S_{t+1}))$	$n = Wbeat$	$s_t > M2$ $\wedge s_{t+1} > M2$	0.0
		$s_t \leq M2$ $\wedge s_{t+1} \leq M2$	1.0
	$n = Dbeat$ $\vee n = Ubeat$	$\forall s \in S$	0.0

Abb. 87 Verbotene Achtelnoten.

B1.11 Beispiel anhand eines 1. Species Counterpoint

Die Bewertung findet immer nur für die vom Nutzer eingegebenen Noten statt. Um die Funktion der Wahrheitstabellen besser visualisieren zu können, werden im Beispiel alle möglichen Noten dargestellt und diese Regel für Regel bewertet. Die Wahrscheinlichkeit wird dabei als grün (sehr wahrscheinlich) über gelb, bis rot (sehr unwahrscheinlich) dargestellt.⁵⁰

Als erste Noten sind nach R1 – die anderen Regeln werden hier nicht angewendet – nur *Perfect Intervals* möglich, wobei die P1 und P8 bevorzugt werden sollten. (Abb. 88)



Abb. 88 Für die erste Note liefert R1 drei Noten mit der Wahrscheinlichkeit 1.0 (grün – C4, C5, C6) und zwei mit der Wahrscheinlichkeit 0.5 (gelb – G4, G5).

Für die zweite Note spielen in diesem Fall nur die Regeln R2, R3, R5 und R6 eine Rolle – das Programm würde natürlich alle Regeln testen.

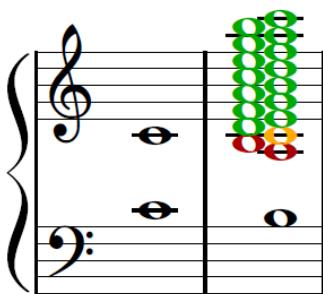


Abb. 91 R2 Bevorzugte Motion

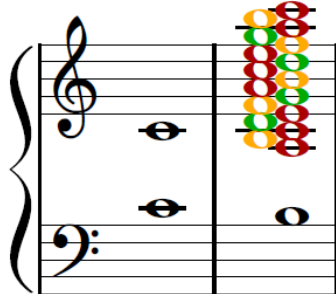


Abb. 91 R3 Bevorzugte Intervalle

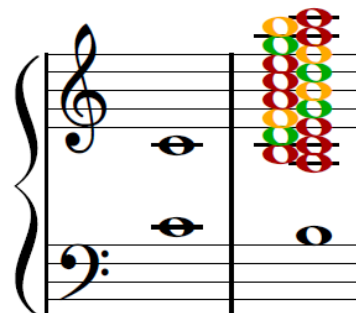


Abb. 91 R2 und R3

R2 hätte nur auf wenige Noten einen Einfluss. R3 würde aber sehr viele Noten ausschließen. Das Produkt der Wahrscheinlichkeiten führt dazu, dass schlechte Noten an Wert verlieren und gute ihre Wahrscheinlichkeit behalten.

⁵⁰ Diese Farben entsprechen nicht exakt der Einfärbung der Noten in CounterPai.

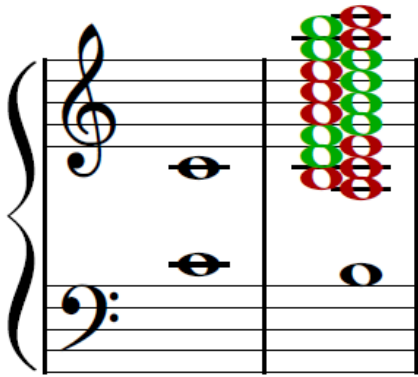


Abb. 93 R5 Bevorzugte Motion-abhängige Intervals.

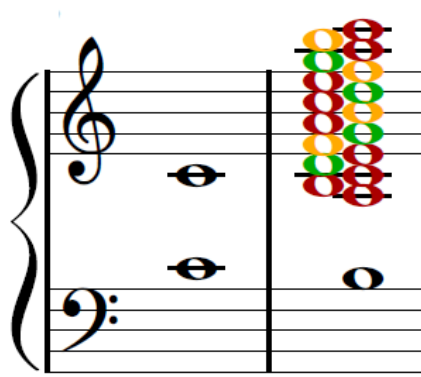


Abb. 93 R2, R3 und R5.

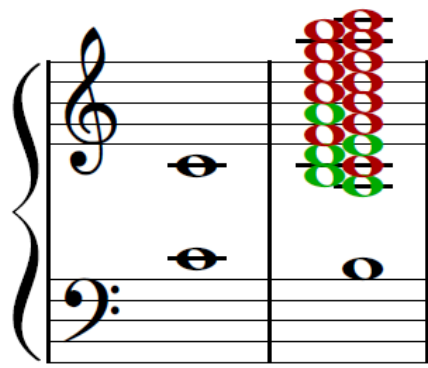


Abb. 95 R6 Verbotene Skips.

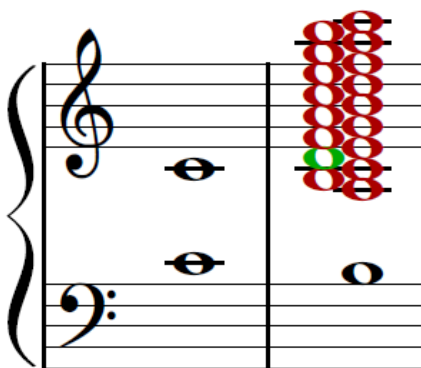


Abb. 95 R2, R3, R5 und R6.

Werden die weiteren Regeln geprüft, ergibt sich, dass in diesem Fall nur noch eine Note einen guten Wahrscheinlichkeitswert liefern würde. (Abb. 95)

Analog zu diesem Beispiel geht der Algorithmus in CounterPai für einzelne Noten vor. Durch das sukzessive Abarbeiten der einzelnen Regeln können den Noten Nachrichten angehängen werden, die die verletzten Regeln dokumentieren.

C Ausschnitt 2 aus der Dokumentation der vorangegangenen Arbeit

Dieser Ausschnitt wurde im Rahmen des vorangegangenen Praxisprojektes [Scholzen, 2019] erstellt und wird hier zum besseren Verständnis für den Leser noch einmal wiedergegeben.

C1 Aufbau von CounterPai

CounterPai unterteilt sich in vier Hauptkomponenten:

Im Mittelpunkt der Komponente Sheet_To_Audio steht die Klasse Sheet_Music mit dem zugehörigen Dateiformat „sheet“. Die Komponente ermöglicht das Abspielen von Sheet_Music-Objekten und das Umwandeln von Midi-Dateien in das sheet-Format.

Die Komponente GUI stellt einen Editor zur Verfügung, mit dem sich Sheet_Music bearbeiten lässt.

Die Komponente Utility ist eine Abstraktionsschicht zwischen CounterPai und externen Bibliotheken.

Die Komponente Evaluator beinhaltet alle Logik, die nötig ist, um Sheet_Music musikalisch zu analysieren und die Kontrapunktregeln zu prüfen. Hier werden im Folgeprojekt auch die AI-Komponenten angesiedelt sein.

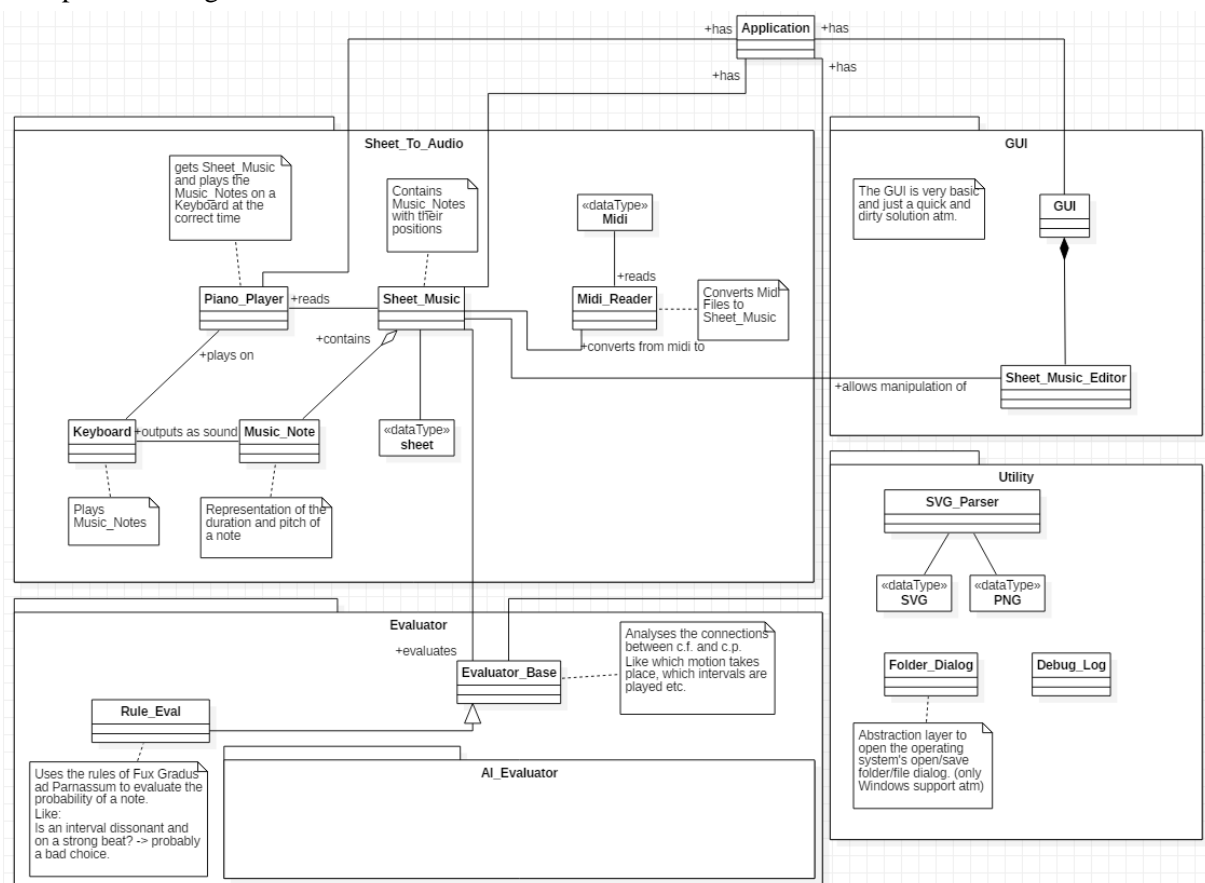


Abb. 96 Vollständiges Klassendiagramm mit Komponentengrenzen. AI Evaluator ist Teil der Folgearbeit und zu diesem Zeitpunkt noch nicht implementiert.

C1.1 Sheet_To_Audio

Die Komponente Sheet_To_Audio ermöglicht den Umgang mit Musikstücken, sie enthält alle Teile zum Abbilden eines Musikstücks, wie Noten und Sheets, sowie die Klassen, die zur Ausgabe der Stücke benötigt werden.

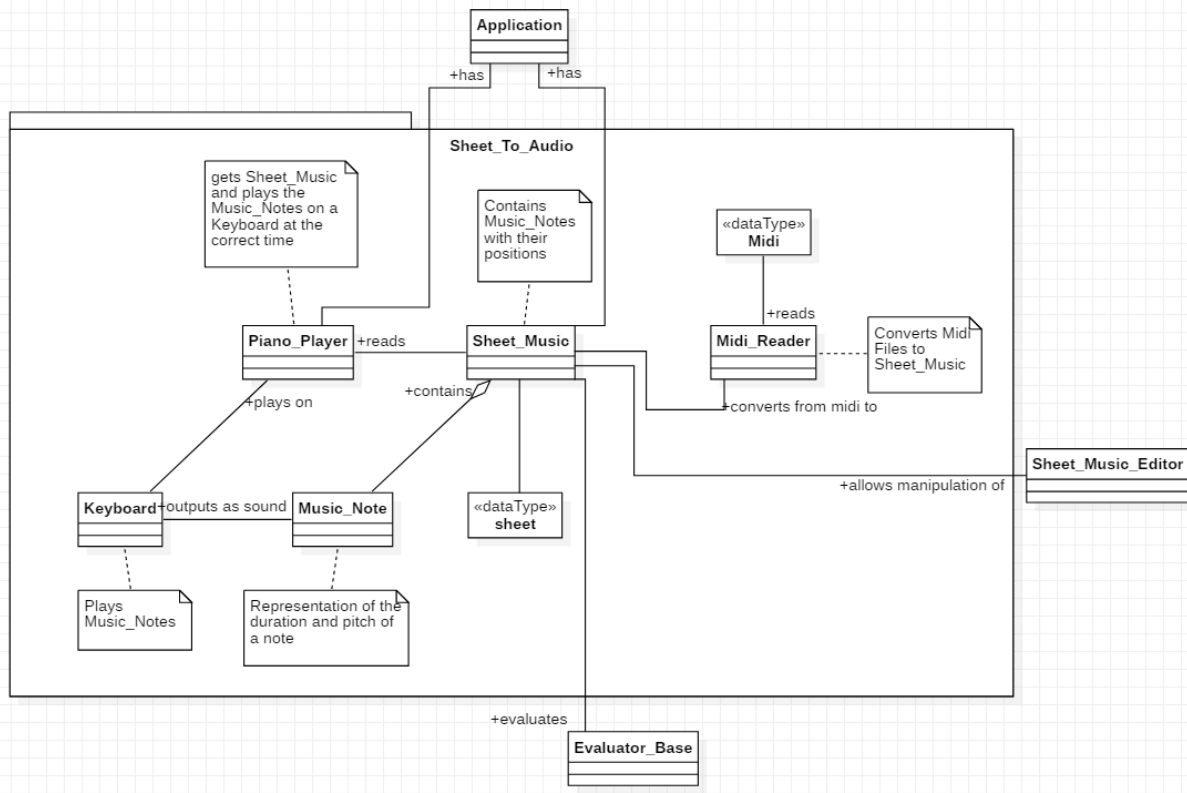


Abb. 97 Klassendiagramm der Komponente Sheet_To_Audio.

Music_Note

Die Music_Note-Klasse besteht aus einem Pitch (z.B. die Tonhöhe C4), einer Dauer (z.B. Whole_Note) und der Information, ob die Note eine gehaltene Note ist. Zusätzlich sind Variablen vorhanden, die Auskunft darüber geben, ob einer Note ein # (*sharp*) oder **b** (*flat*) vorgezeichnet ist. Außerdem wird die Stimme (*Voice*), zu der die Note gehört, gespeichert. Diese Information ist eigentlich durch die Sheet_Music gegeben, eine zusätzliche Speicherung erleichtert aber den Umgang mit den Noten in den Evaluator-Klassen.

Jeder Music_Note kann eine Nachricht als json-formatierter String angehängt werden⁵¹. In den sheet-Dateien Abb. 98 enthalten die Noten nur leere Nachrichten-Strings.

```
tempo:
200
cf:
bass
soprano:
A4 p: 69 va: 2 vo: 1 f: 0 s: 0 t: 1 | {}
A4 p: 69 va: 2 vo: 1 f: 0 s: 0 t: 0 | {}
:
D5 p: 74 va: 1 vo: 1 f: 0 s: 0 t: 0 | {}
bass:
D3 p: 50 va: 1 vo: 0 f: 0 s: 0 t: 0 | {}
F3 p: 53 va: 1 vo: 0 f: 0 s: 0 t: 0 | {}
:
D3 p: 50 va: 1 vo: 0 f: 0 s: 0 t: 0 | {}
end
```

Abb. 98 Ausschnitt einer sheet-Datei. (p: Pitch, va: Notenwert, vo: Voice, f: Flat, s: Sharp, t: Tied)

⁵¹ Der Nachrichtenstring ermöglicht es, die Komponente Sheet_To_Audio komplett von den anderen Komponenten zu trennen. So können andere Komponenten den Noten weitere Eigenschaften anhängen, ohne die Klasse Note oder das sheet-Format ändern zu müssen.

Sheet_Music

Die Klasse `Sheet_Music` besteht aus zwei *Voices*: dem Bass und dem Soprano. Die *Voices* sind als *Linked Lists* aus `Music_Note`-Objekten implementiert. Die Klasse `Sheet_Music` stellt Funktionen zur Verfügung, diese Listen zu manipulieren.

Weiter ermöglicht `Sheet_Music` das Abspeichern und Laden von sheet-Dateien. Für das Dateiformat wurde in Betracht gezogen, die `Sheet_Music`-Objekte einfach als Bitstrom zu speichern, im Hinblick auf Erleichterung des Debugging wurde aber schließlich ein lesbares Format gewählt. (Abb. 98)

Die *Voice*-Manipulationsfunktionen sind keine einfachen Listenoptionen, wie *add* oder *pop_last*, sondern orientieren sich immer an den Taktstrichen und den schon in der Liste vorhandenen Noten.

Die Funktion `add_note(Music_Note, sixteenth_distance)` fügt eine Note an einer bestimmten Stelle hinzu. Diese Stelle wird als Sechzehntelnoten vom Beginn des Sheets angegeben. (Abb. 99)

Der Vorgang unterteilt sich in drei Schritte: das Suchen nach der richtigen Position in der Liste, das Einfügen (bzw. Ändern) einer Note und das anschließende Säubern. Das Säubern teilt Noten, die einen Taktstrich überschreiten würden, in gehaltene Noten.

Zuerst wird die Position der neuen Note ermittelt, diese Position führt zu einem der drei Fälle:

```
WAS_BETWEEN_NOTES,
WAS_AT_NOTE,
WAS_AFTER_LAST_NOTE
```

`WAS_BETWEEN_NOTES` bedeutet, dass eine Note zwischen zwei schon vorhandenen Noten eingefügt werden muss. `WAS_AT_NOTE` bedeutet, dass an dieser Stelle schon eine Note vorhanden ist und diese nur geändert werden muss. `WAS_AFTER_LAST_NOTE` bedeutet, dass die Note hinter der letzten Note eingefügt werden muss.

Wird die Note zwischen zwei Noten eingefügt, muss die Vorgängernote entsprechend der Position der neuen Note verkürzt werden.

Unter Umständen führt dies auch dazu, dass die Vorgängernote in mehrere gehaltene Noten aufgeteilt werden muss.

Anschließend wird ein Clean-Up durchgeführt, bei dem Noten, die über einen Taktstrich hinweg gehen, geteilt werden.



Abb. 99 Der Funktion `add_note(...)` wird als Input ein `C5` als Viertelnote und die Distanz 26 Sechzehntel übergeben. Die ganze Note `B4` wird zu einer halben verbunden mit einer Viertelnote und die neue Note wird eingefügt.

Keyboard

Das Keyboard ermöglicht es, Noten abzuspielen. Dabei verwendet es mehrere WAV-Dateien, die im Vorfeld von einem Klavier abgenommen wurden. Jede WAV-Datei beinhaltet den Ton einer der 88 möglichen Klaviertasten.

Die Funktionalität der Keyboardklasse wurde der eines realen Klaviers nachempfunden. Die Tasten können entweder gespielt oder losgelassen werden. Solange eine Taste noch gehalten wird, entsteht kein neuer Ton, wenn sie erneut gedrückt wird. Wann und wie lange eine Taste gedrückt wird, wird durch die Klasse `Piano_Player` festgelegt.

Piano_Player

Der `Piano_Player` kann ein `Sheet_Music` Objekt lesen und auf einem Keyboard abspielen. Der `Piano_Player` füllt eine *Priority Queue* mit Events, die die `Sheet_Music` repräsentieren. Ein Event besteht aus der zu spielenden Note, dem Event (PLAY oder STOP) und dem Zeitpunkt, an dem das Event stattfindet. Der früheste Zeitpunkt hat in der *Queue* die höchste Priorität.

Soll zum Beispiel C4 eine ganze Note gespielt werden, das Tempo ist 60bpm (bezogen auf Viertelnoten) und es ist 12:00:00 Uhr, wird der *Queue* ein Event „PLAY, C4, 12:00:00“ und ein Event „STOP, C4, 12:00:04“ hinzugefügt.

Für den `Piano_Player` wird aus der Application zu jedem Frame eine *update()*-Funktion aufgerufen. Bei diesem Update werden alle Events der *Queue* ausgeführt, deren Zeitpunkt vor der aktuellen Systemzeit liegen.

Midi_Reader

Der Midi_Reader erlaubt es, midi-Dateien in Sheet_Music Objekte umzuwandeln. Da Midi-Dateien Midi-Events beinhalten (diese sind den Events des Piano_Players sehr ähnlich), wurden einige Einschränkungen für die importierbaren Midi-Dateien festgelegt. Es können nur Dateien gelesen werden, die bei 100bpm eingespielt, bzw. eingegeben wurden. Außerdem können nur Noten, die nicht kleiner als Achtelnoten sind, vom Parser erkannt werden. Zusätzlich müssen die Dateien zwei Voices enthalten, die zu jeder Zeit einen Ton spielen (also keine langen Pausen enthalten).⁵²

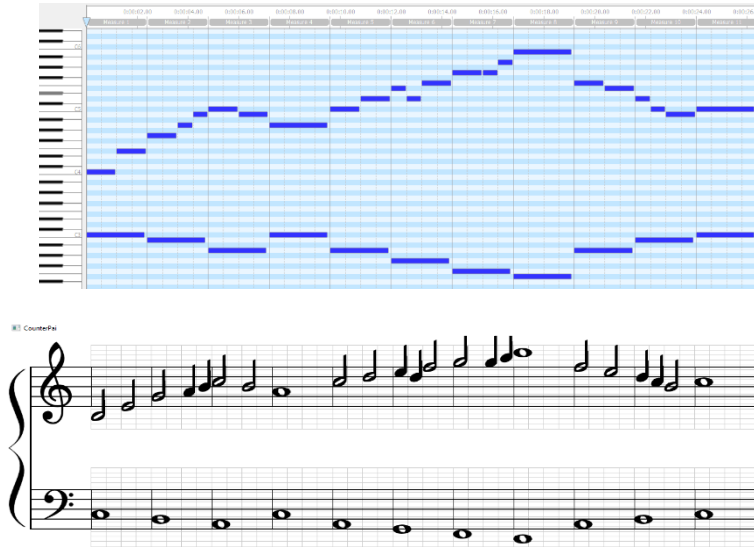


Abb. 100 Oben ein Midi-File angezeigt als Zeitleiste. Man kann in der Basslinie erkennen, dass die Töne nicht durchgängig spielen. Unten das Ergebnis des Imports.

Für den Import wurde ein Algorithmus entworfen, der eine Midi-Datei einliest und dann in das sheet-Format umwandelt. Dieser Algorithmus prüft für jedes Midi-Event, ob während des Events noch andere stattfinden. Finden andere Events statt, wird wiederum geprüft, welches Event die Note mit dem höheren Pitch erzeugt. Auf diese Weise wird festgelegt, ob die Note dem Soprano oder dem Bass hinzugefügt werden soll.

Für den Import wird die Länge des Events betrachtet und absteigend von der größtmöglichen Notendauer ($dur_N := \text{Dauer der infrage kommenden Note}$) bis zur Kleinsten geprüft, ob die Eventdauer ($dur_E := \text{Dauer des Events}$) übereinstimmt. Da die Events nicht exakt sind, wird eine Abweichung $\Delta d := \text{Abweichung}$ berücksichtigt:

$$abs(dur_E - dur_N) \leq \frac{dur_N}{\Delta d}$$

Für $\Delta d = 5.0$ konnten die besten Ergebnisse erzielt werden.

Für eine ganze Note ergäbe sich bei 100bpm (bezogen auf Viertelnoten) der Wert $dur_N = 2.4$. Ein Event im Wertebereich $1.92 \leq dur_E \leq 2.88$ würde also als ganze Note erkannt.

⁵² Midi wurde als ein Liveübertragungsformat entworfen. Es bietet die Möglichkeit, auf 128 unterschiedlichen Channels Informationen zu senden. Bei Liveauftritten könnte zum Beispiel ein Keyboard auf Channel 1 senden und ein digitales Schlagzeug auf Channel 2. Ein Programm würde für die verschiedenen Channels dann entscheiden, was mit den gesendeten Informationen geschieht. Diese Channels werden in einigen Datenbanken klassischer Musik dafür verwendet, die unterschiedlichen Stimmen abzuspeichern.

C1.2 GUI

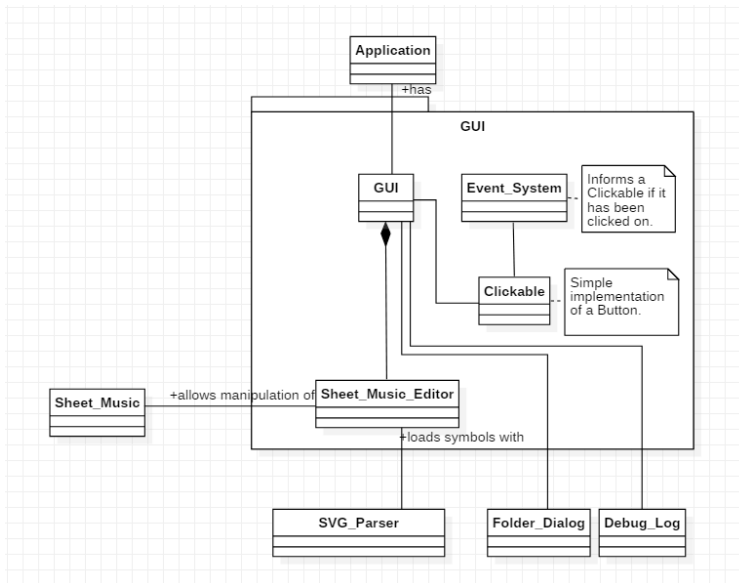


Abb. 101 Klassendiagramm der Komponente GUI.

Die Komponente GUI⁵³ stellt alle Funktionen bereit, um dem User den Umgang mit CounterPai zu ermöglichen.

GUI (Klasse)

Die Klasse GUI beinhaltet alle Objekte, die mit dem User In-/Output zu tun haben.

Hier werden alle Buttonpositionen und deren Funktionen festgelegt.

Clickable

Clickable stellt eine Grundlage für einen Button dar. Clickables besitzen

eine Funktion `on_clicked()`, die per Lambdaausdruck⁵⁴ gesetzt werden kann. Clickables beinhalten außerdem ein Rechteck (Clickarea), das angibt, an welche Stelle geklickt werden muss, damit das Clickable aktiviert wird.

Event_System

Für das Event_System wurde ein Singletonpattern⁵⁵ gewählt. Es interagiert nur mit Clickables, diese werden per Publish-Subscribe-Methode verwaltet:

Clickables registrieren sich bei ihrer Erzeugung beim Event_System. Stellt das Eventsystem fest, dass die Maus im Clickarea eines Clickables geklickt wurde, ruft das Event_System die `on_clicked()`-Funktion des Clickables auf.

Sheet_Music_Editor

Der Editor ermöglicht die Eingabe und Anzeige von Sheet_Music. Er stellt ein *Grid* bereit, das beim Klicken in ein Feld das angezeigte Sheet_Music-Objekt manipuliert. Im Editor befinden sich Optionen wie „is_tying“ oder „is_deleting“, die beeinflussen, welche Aktionen auf das Sheet_Music-Objekt angewendet werden. Diese Optionen werden durch die GUI-Buttons gesetzt.

⁵³ Zu Beginn des Projekts wurde eine GUI-API nicht für nötig erachtet. Im Rückblick ist festzuhalten, dass dies eine Fehlentscheidung war. Schon simple Funktionen wie das Öffnen eines Ordnerdialogs stellen einen erheblichen Programmieraufwand dar.

⁵⁴ Eine namenlose Funktion, die überall im Code definiert werden kann.

⁵⁵ Eine Programmieretechnik, die sicherstellt, dass von einer Klasse nur ein einziges Objekt erstellt werden kann und die es erleichtert, dieses Objekt von jeder Stelle im Code aufrufen zu können.

C1.3 Utility

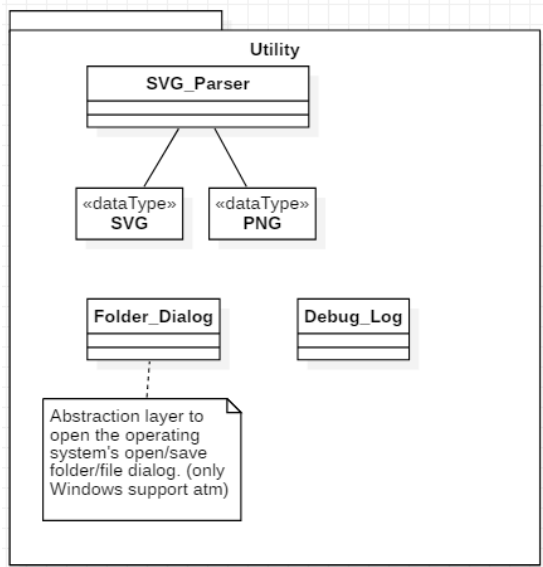


Abb. 102 Die Komponente Utility

Die Komponente Utility beinhaltet Funktionen, die im gesamten Programm hilfreich sein könnten, aber die vom Umfang zu klein sind, um eine eigene Komponente zu rechtfertigen.

SVG_Parser

Der SVG_Parser wandelt SVG-Dateien in das durch SFML lesbare PNG-Format um. Er wird verwendet, um die Notensymbole und die Notenschlüssel zu laden.

Folder_Dialog

Folder_Dialog ist eine Abstraktionsschicht, die den Umgang mit betriebssystemspezifischen Dateisystemen ermöglicht. Sie stellt Funktionen bereit, um die Position der ausgeführten Datei

festzustellen und einen Ordnerdialog zu öffnen. Bisher ist allerdings nur der Windows-Ordnerdialog implementiert.

Debug_Log

Der Debug_Log loggt – wie der gewiefte Leser vermutlich schon erraten kann – Debugnachrichten in einem Log. Hauptsächlich wird die Logfunktion von der GUI-Klasse aufgerufen, um Userinput, der zu einem Absturz geführt hat, rekonstruieren zu können.

C1.4 Evaluator

Die Evaluator-Komponente beinhaltet die Logik hinter den Kontrapunktregeln nach Fux. Jeder Evaluator erbt von der Evaluator_Base-Klasse. Diese beinhaltet Funktionen, um eine allgemeine Musikanalyse der Noten durchzuführen.

Die erbende Klasse Rule_Eval nutzt die von der Basisklasse zur Verfügung gestellte Analyse und bewertet mit ihr die Noten nach den Kontrapunktregeln.

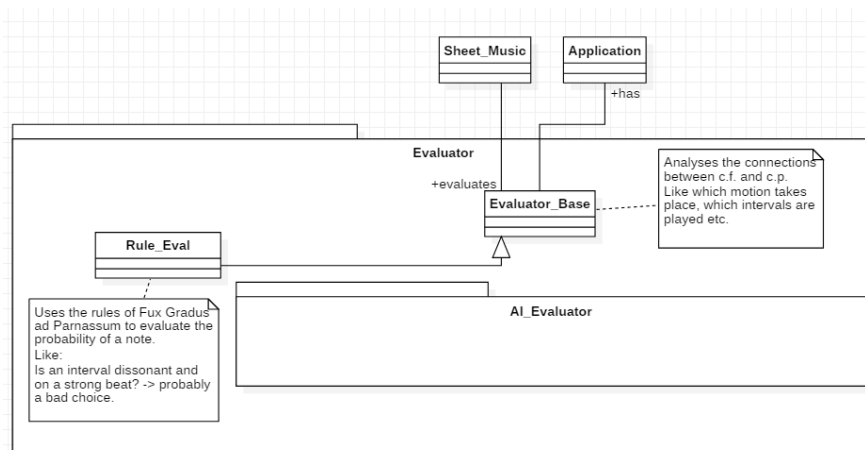


Abb. 103 Die Komponente Evaluator

Evaluator_Base

Einem Evaluator_Base-Objekt kann ein Sheet_Music-Objekt zur Analyse übergeben werden.

Die Beziehungen zwischen den einzelnen Noten werden dann in einer Note_Evaluation zusammengefasst.

Zu jeder Note im *c.p.*⁵⁶ wird eine

Note_Evaluation angelegt, die alle Beziehungen dieser Note speichert. Diese Beziehungen können zum Beispiel sein, dass die *c.p.*-Note ein Intervall einer Oktave mit dem *c.f.* bildet oder dass die Note mit einer Aufwärtsbewegung erzeugt wurde. Diese Informationen werden den Music_Notes anschließend angefügt. Das geschieht, indem die Evaluator_Base sich mit dem Namen „NOTE_EVALUATION“ bei den Noten registriert und dann die Nachricht hinterlegt. (Abb. 104)

Rule_Evaluator

Die eigentlichen Regeln des Kontrapunkts werden in der Komponente Rule_Evaluator geprüft. Alle Note_Evaluations, die durch die Basisklasse zur Verfügung gestellt werden, werden auf alle möglichen Regeln geprüft. Jede verletzte Regel führt zu einer niedrigeren Wahrscheinlichkeit der Music_Note auf die sich die Note_Evaluation bezieht. Zusätzlich wird den Noten angefügt, welche Regeln verletzt wurden und für wie wahrscheinlich der Rule_Evaluator die Note hält. Die Nachricht registriert der

```

E5 p: 76 va: 4 vo: 1 f: 0 s: 0 t: 1 | {"NOTE_EVALUATION": "p: Mid_Bar b: Down_Beat j: P1 d: Side m: ObMo i: M7", "RULE_EVALUATION": "prb: 0.45 R2 R3"}
  
```

Abb. 104 Eine Note im sheet-Format mit angefügten Nachrichten. (NOTE_EVALUATION: p: Bar-Position, b: Beat-Position, j: Jump-Interval, d: Direction, m: Motion, i: Interval mit dem c.f. RULE_EVALUATION: prb: probability, R2, R3: Regeln 2 und 3 wurden verletzt.

Rule_Evaluator mit dem Namen „RULE_EVALUATION“. (Abb. 104)

⁵⁶ Der c.p. (counter point) und der c.f. (cantus firmus) sind die zwei Stimmen im Kontrapunkt.

Eidesstattliche Erklärung

Ich versichere an Eides Statt, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum

Unterschrift