Hochschule Düsseldorf
University of Applied Sciences
HSD

Fachbereich Medien
Faculty of Media

WISSENSCHAFTLICHE VERTIEFUNG

Autmoatic Music Transcription

Autor:
Benedikt Kolodziej
878007
benedikt.kolodziej@study.hs-duesseldorf.de
Medieninformatik (B. Sc.)

Betreuender Professor:
Prof. Dr. Dennis Müller
dennis.mueller@hs-duesseldorf.de

Zeitraum
28.05.2025 - xy

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Automatic Music Transcription . . . 1
- 1.2 Herausforderungen und Hindernisse . . . 1
- 1.3 AMT und Künstliche Intelligenz, . . . 1
- 1.4 praktische Anwendungsfelder und Vorteile von AMT . . . 3
- 1.5 Motivation und Zielsetzung dieser Arbeit . . . 3

2 Geschichtliche Entwickelung von AMT-Systemen **3**
- 2.1 Moorer und eines der ersten Musiktranskriptionssysteme . . . 3
  - 2.1.1 Grundlegende Probleme bei AMTs . . . 3
  - 2.1.2 Der Aufbau von Moorers AMT-Systems . . . 4
  - 2.1.3 Hidden Markov Models . . . 5
- 2.2 MIDI-Dateien . . . 5
- 2.3 Entwickelung von Datensätzen . . . 6
- 2.4 Polyphone AMT-Systeme und neue Ansätze . . . 7
  - 2.4.1 Blackboard-System . . . 7
  - 2.4.2 RASTA-basiertes System . . . 9
- 2.5 Einbindung von Künstlicher Intelligenz . . . 10

3 Hindernisse Moderner AMT-Systeme **10**
- 3.1 Onset Detection . . . 11
- 3.2 Polyphonie und Notenzuordnung . . . 11
- 3.3 Instrument spezifische Probleme . . . 11
- 3.4 Notendauer . . . 12
- 3.5 Reale Audioaufnahmen . . . 12
- 3.6 Frame-basierte vs Event-basierte AMT-Systeme . . . 12
- 3.7 Blackbox von KI-Modellen . . . 13

4 KI-Module in AMT-Systemen **13**
- 4.1 Convolutional Neural Networks . . . 13
  - 4.1.1 Layer eines CNNs . . . 14
  - 4.1.2 Geschichte von CNNs . . . 14
- 4.2 Recurrent Neural Networks . . . 15
  - 4.2.1 Grundstruktur eines RNNs . . . 15
  - 4.2.2 Aufgaben eines RNNs . . . 16
  - 4.2.3 Long Short-Term Memory . . . 17
  - 4.2.4 Gated Recurrent Units . . . 17
  - 4.2.5 Bidirektionale RNNs . . . 18
  - 4.2.6 Die Geschichte von RNNs . . . 18
- 4.3 Transformers . . . 18
  - 4.3.1 Input Embedding und Position Encoding . . . 18
  - 4.3.2 Self-Attention Layer . . . 19
  - 4.3.3 Feedforward Layer . . . 19
  - 4.3.4 Geschichte . . . 20
- 4.4 Weitere Deep-Learning-Module und Entwicklungen . . . 20
  - 4.4.1 Variational Autoencoder . . . 20
  - 4.4.2 Graph Neural Network . . . 21
  - 4.4.3 Diffusion Model . . . 21
  - 4.4.4 Reinforcement Learning . . . 21
  - 4.4.5 Energy-Based Model . . . 21

5 KI-basierende AMT-Systeme im Vergleich **22**

5.1 Omnizart 22
5.2 MT3 24
5.3 Omnizart vs MT3 25

6 Fazit **26**

Literaturverzeichnis **27**

Abbildungsverzeichnis

1 Modifizierter Ausschnitt aus [34] 2
2 Systemstruktur des AMT-Ansatzes von Keith D. Martin (1996), basierend auf [32]. . 8

1 Einleitung

1.1 Automatic Music Transcription

Musik ist seit Jahrtausenden ein zentraler Bestandteil unserer Gesellschaft. Während etwa 2000 bis 0 v.Chr. musikalische Werke meist mündlich überliefert wurden, entwickelte sich in diesem Zeitraum auch eine Notenschrift. Diese Notenschrift ermöglichte es, Musikstücke einfacher zu erlernen und einem breiteren Publikum zugänglich zu machen. Durch die Digitalisierung erhielten Digital Audio Workstations zunehmend Einzug in die Musikproduktion, wodurch Notenblätter oft nicht mehr notwendig waren und es weniger Bedarf gab diese Lieder zu übersetzen in Notenschrift.

An dieser Stelle setzt Automatic Music Transcription (AMT) an. AMT ist ein Prozess, bei dem eine Audiospur als Input gegeben wird und diese durch Computerprogramme Notenblätter oder, was weiter verbreitet ist, MIDI-Dateien als Output wiedergeben. Dabei werden durch mehrere Prozesse die Eigenschaften der Noten, zum Beispiel Frequenz oder Lautstärke, analysiert und im Kontext des Musikstückes analysiert.

1.2 Herausforderungen und Hindernisse

Anstatt das man selber diese Lieder, alleine durchs Gehör, in Notenschrift überträgt würde diese Aufgabe eine Software für einen erledigen. Dieses Ziel ist jedoch schwer zu erreichen, da Musik mehrdimensional ist durch zum Beispiel Zeit, Tonhöhe und Polyphonie. Vor allem bei polyphonen Musikstücken haben herkömmliche Algorithmen viele Schwierigkeiten. In diesen Fällen müssen sie nämlich viele verschiedene Stimmen gleichzeitig analysieren und im späteren auch die jeweiligen Töne voneinander differenzieren und eindeutig einem Instrument zuordnen. Ein weiteres Problem ist die Individualität jedes Musikstückes. In realen Aufnahmen können leichtes Rauschen, kleine Spielfehler oder stilistische Mittel wie Vibrato auftreten, die je nach Interpreten unterschiedlich klingen. Zudem sind die meisten AMT-Modelle auf westliche Tonleiter trainiert. Dies kann zu Problemen führen, wenn man zum Beispiel arabische oder indische Musikstücke transkribieren möchte.

1.3 AMT und Künstliche Intelligenz,

Um diese Vielfalt zu bewältigen, ist ein neuer, oft genutzter Ansatz, die Nutzung von künstlicher Intelligenz und Machine Learning. Im Gegensatz zu Algorithmen ist KI flexibler und kann sich besser einstellen auf kleine Abweichungen in Musikstücken. Reale Audiospuren besitzen immer eine gewisse Menge Rauschen. Das kann bei der automatischen Musiktranskription für schwierigkeiten sorgen, da Algorithmen diese als zusätzliche Noten ansehen oder richtige Noten dadurch nicht erkennen könnten. KIs können sich besser an diese begebenheiten anpassen, da neuronale Netzwerke mit genau diesen unperfekten Audiosignalen trainiert werden können. Somit können AMT-Systeme besser angepasst werden für einen realistischen Gebrauch.

Auch die Mehrdimensionalität von Musik kann KI deutlich besser bewältigen als Algorithmen. Neuronale Netze besitzen eine mehrdimensionale Struktur, die es ihnen ermöglicht, verschiedene Muster, Stimmen und Eigenschaften zu erlernen. [12] Auf der anderen Seite müssen klassische Algorithmen diese verschiedenen Dimensionen explizit modellieren und sind nicht in der Lage, Muster selbstständig zu erkennen. Sie folgen nur dem, was zuvor vom Menschen fest einprogrammiert wurde.

In der automatischen Musiktranskription nutzt man meistens Spektrogramme, zur darstellung der Audiodatei. Spektrogramme zeigen den zeitlichen Verlauf des Freqeunzspektrums eines Audiosignals. Es gibt verschiedene Arten von Spektrogrammen. Zwei Spektrogrammtypen, die häufig genutzt werden, sind CQT-Spektrogramme und Log-Mel-Spektrogramme. CQT steht für Constant-Q Transform und in diesem Spektrogramm werden die Frequenzachsen logarithmisch aufgelöst. Zudem bleibt der Q-Faktor konstant, dieser beschreibt das Verhältnis von Frequenz zu Brandbreite. Das Log-Mel-Spektrogramm wird zunächst mit einer Short-Time Fourier Transform (STFT) gebildet. STFT ist eine Methode, um ein Signal als ein Frequenzspektrum darzustellen. Anschließend wird die lineare Frequenzachse auf eine Mel-Skala projiziert. Das menschliche Gehör kann 200–400Hz feiner als 5000–

5200Hz wahrnehmen. Die Mel-Skala sorgt dafür, das sich das Spektrogramm dieser menschlichen Hörweise anpasst. Insgesamt ist ein CQT-Spektrogramm besser für die automatische Musiktranskription, da dieses Töne feiner unterscheiden kann. Jedoch werden in vielen neueren KI-Modellen eher Log-Mel-Spektrogramme benutzt, da diese robuster sind und vor allem besser in Transformer-basierte KI-Modellen laufen. Im folgenden

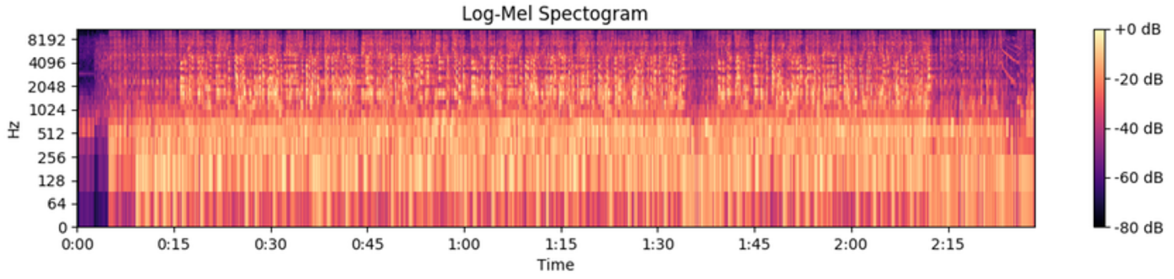


Abbildung 1: Modifizierter Ausschnitt aus [34]

Um ein AMT-Modell mit KI zu kreieren, muss man sich auch für ein KI-Modell entscheiden. Hier werden meistens Recurrent Neural Networks (RNN) oder Convolutional Neural Networks (CNN), als einzelne Module, benutzt. [2] Diese Module bilden keine eigenständigen Systeme, sondern lassen sich flexibel innerhalb eines Systems kombinieren. Da es keine objektiv richtigen oder falschen Notenfolgen gibt, ist der Einsatz von Reinforcement Learning nicht sinnvoll. Dementsprechend braucht man auch ein zuverlässiges Datenset aus Audiodateien und deren zugehörigen MIDI-Dateien.

CNNs können gut räumliche Strukturen erkennen. Das hiflt uns bei der Analyse von Spektrogrammen. Meist werden die verschiedenen Frequenzen der Noten, die gespielt wurden, nach der Verarbeitung des Musikstückes in Spektrogrammen wiedergegeben. Durch die Analyse von dem Spektrogramm können gewisse Frequenzmuster erkannt werden, die dann einem bestimmten Instrument zugeordnet werden können. Das ist vor allem hilfreich dabei verschiedene Stimmen der jeweiligen Instrumente voneinander zu differenzieren. [15]

RNNs hingegen sind spezialisiert, um zeitliche Abläufe besser im Kontext zu verstehen. In der Musik werden viele Noten hintereinander gespielt, diese müssen harmonisch im Stück übereinstimmen. Das RNN verarbeitet die jeweiligen Sequenzen und merkt sich die Informationen der schon gespielten Noten, um die darauffolgenden Noten besser einordnen zu können. So lassen sich Tonfolgen harmonischen Strukturen zuordnen oder der Rhythmus des Stücks analysieren. [2]

RNNs und CNNs werden auch häufig kombiniert in AMT-Modellen. Meist in folgender Reihenfolge:

1. **CNN:** Extrahiert folgende Merkmale aus dem Spektrogramm:
    - Frequenzverteilungen und spektrale Muster
    - Tonhöhenlage und damit verbundene Obertöne
    - Klangfarbe einzelner Instrumente
    - Energieverteilung, unter anderem zur Erkennung von Toneinsätzen
    - Harmonische Strukturen wie Akkordfolgen
2. **RNN:** Verarbeitet auf Basis dieser Merkmale die zeitliche Abfolge und erkennt dabei folgende Eigenschaften:
    - Reihenfolge und Übergänge musikalischer Ereignisse
    - Beginn und Ende einzelner Töne zur Bestimmung der Notendauer
    - Rhythmische Muster und zeitliche Gruppierungen
    - Musikalische Phrasen mit zusammenhängender Struktur

- Wiederholungen, Themen oder längere Abhängigkeiten im Verlauf

3. **Output:** Gibt das transkribierte Musikstück in strukturierter Form aus:
    - Als MIDI-Datei mit exakten Noteninformationen

Nachdem man die KI diesen Prozess durchlaufen lassen hat, kann man mit einem eigenen oder externen Programm diese MIDI-Dateien zu standardisierter Notenschrift transkribieren.

Es gibt aber auch Projekte, wo nur ein bestimmter Teil dieser Kette mit KI-Modulen realisiert wird, oder andere KI-Module verwendet werden. Mehr zu diesen Modulen und CNNs und RNNs im Detail erläutere ich in dem Abschnitt-**??**.

1.4 praktische Anwendungsfelder und Vorteile von AMT

AMT kann auch bei vielen anderen Problemen helfen oder in vielen Bereichen Quality-of-Life-Changes bringen. Zum einen kann der Musikunterricht spannender und interaktiver gestaltet werden. Es gibt eine breitere Auswahl von Musikstücken, die man den Schülern anbieten kann, wodurch diese durch individuell angepasste Musikstücke mehr Spaß und Ehrgeiz beim lernen haben könnten. Zudem kann man die gespielten Musikstücke der Schüler direkt beim Spielen transkribieren und gezielt erkennen, wo der jeweilige Schüler noch Verbesserungsmöglichkeiten hat. Grundsätzlich können deutlich mehr Musikstücke transkribiert werden, wodurch sich große Archive aufbauen lassen. Ein größeres Interesse an Musik wird geweckt, da Musikstücke von beliebten Serien, Filmen oder Spielen leichter für deren Musikbegeisterte Zielgruppe zugänglich sind. Alleine dadurch, dass Computerprogramme Musikstücke besser verstehen, können darauf aufbauend weitere Tools für die Musikproduktion entwickelt werden. Auch KI würde davon stark profitieren. KI-generierte Musik würde verbessert werden, da die KI selber ein besseres Verständnis der Musik entwickelt. Audiobasierte Suchmaschinen könnten gewünschte Musikstücke oder bestimmte Videos präziser finden. Musik könnte barrierefreier gestaltet werden, indem gehörlose Menschen sie lesen können und Musiker beim Spielen direktes Feedback erhalten, ob sie die Noten korrekt gespielt haben.

1.5 Motivation und Zielsetzung dieser Arbeit

(Noch nicht angefangen!) Was wird in der Arbeit behandelt, worauf liegt der Fokus (z.B. KI-Methoden für AMT), warum ist das Thema relevant (z.B. für Musiker, KI-Forschung, Musikpädagogik)?

2 Geschichtliche Entwickelung von AMT-Systemen

2.1 Moorer und eines der ersten Musiktranskriptionssysteme

2.1.1 Grundlegende Probleme bei AMTs

Einer der ersten Papers über Automatic Music Transcription wurde im Jahr 1977 geschrieben. [33] In dieser beschreibt Moorer seinen Ansatz, polyphone Musik Audiospuren direkt über Computerprogramme in Notenschrift zu übertragen. Während dieses Prozesses fallen ihm schon sehr viele schwierigkeiten auf, die auch in späteren Papern und Arbeiten eine ausschlaggebende Rolle spielen werden.

Eines dieser Probleme wird von ihm als das "Cocktail-Party-Problem" bezeichnet. Dieses stellt die Schwierigkeit dar, auf einer Party bestimmten Stimmen zu folgen, während viele verschiedene Stimmen gleichzeitig erklingen. Das gleiche Problem liegt in der Noten transkription. Die meisten Musikstücke haben mehrere Instrumente, welche gleichzeitig spielen. Öfter gibt es auch Musikstücke wo es für ein Instrument, wie zum Beispiel Violine, mehrere verschiedene Stimmen gibt. Dies erschwert die Zuordnung bestimmter Noten zu einer gewählten Stimme. Schon viele Menschen scheitern deshalb daran große Musikstücke richtig zu transkribieren. Noch schwieriger wird es hier für Computerprogramme. Anfangs identifizieren diese bestimmte Töne anhand der Frequenz des Tons. Leider reicht

das, wie Moorer feststellt, nicht aus um genaustens zu bestimmen, welche Töne genau momentan gespielt werden.

Jeder Ton hat Obertöne. Diese Obertöne sind jeweils das Vielfache von dem Grundton, den man spielt. Heißt, wenn ich auf einem Klavier den Ton C3 mit 130,81 Hz spiele dann hat dieser die Obertöne C4 (261,62 Hz), G4 (392,42 Hz) usw. Wenn man nur C3 spielt erklingen für das Computerprogramm auch die jeweiligen Obertöne, was man Frequenzüberlagerung nennt. Diese Frequenzüberlagerung sorgt dafür, das zum Beispiel ein Klavier anders klingt als eine Violine. Daraus resultiert dann die Klangfarbe (Timbre) eines bestimmten Instrumentes. [11] Leider konnte Moorer zu dieser Zeit noch nicht die Klangfarbe eines Instruments erkennen. Er konnte auch noch nicht das Problem der Obertöne lösen, da die damaligen Algorithmen und Verfahren noch nicht in der Lage waren die Grundfrequenz von den Obertönen zu trennen, weshalb er sich ausschließlich auf zweistimmige polyphone Musikstücke fokussierte.

Ein weiteres Problem war Rauschen in realistischen Audiospuren und Stilistische mittel in der Musik, wie zum Beispiel Vibrato. In real aufgenommenen Audiospuren gibt es immer ein gewisses Hintergrundrauschen. [18] Dieses kann von einem Computerprogramm auch als Note erkannt werden oder verhindern, das bestimmte Noten richtig vom Computerprogramm erkannt werden. Moorer hat ein Musikstück analog aufgenommen und dieses dann mit einem 14-Bit converter digitalisiert. Dadurch war das Rauschen nicht weg, aber da er das Musikstück selber aufgenommen hat und dieses konvertiert hat sorgte es für insgesamt geringeres Rauschen. Zudem konnte er so die Musiker davon abhalten bestimmte Stilistische mittel zu verwenden, um bessere Daten zur Transkription zu erhalten. Stilistische mittel, wie Vibrato, konnten nicht genutzt werden, da diese eine kleine aber kontinuierliche Veränderung der Frequenz verursachen. Dadurch kann das Computerprogramm nicht korrekt erkennen, das eigentlich eine einzelne Note gespielt wurde. Somit war der Onset und Offset der Note komplett falsch.

Das letzte Problem, was Moorer angesprochen hat, ist das Nutzen von nicht harmonischen Instrumenten wie Trommeln oder einem Schlagzeug. Diese Instrumente haben keinen eindeutigen Pitch für deren Töne, sie sind eher abhängig von Rhythms und Lautstärke. Da Moorers AMT sich jedoch auf das Frequenzmuster der Noten fokussiert, können diese Musikinstrumente nicht berücksichtigt werden.

2.1.2 Der Aufbau von Moorers AMT-Systems

Moorers automatische Musiktranskriptionssystem war eins der ersten seiner Art. Viele weiteren AMT-System leiten sich von diesen ab.

Zunächst wird ein analoges Musiksignal mit einem 14-Bit converter digitalisiert. Dieses digitale Musiksignal wurden dann genutzt um mithilfe von Bandpassfiltern, ein Filter welcher nur bestimmte Frequenzen durchlässt, bestimmte Frequenzbereiche zu isolieren. Dadurch konnte Moorer die gespielte Note und deren Dauer, also zugleich auch deren Onset und Offset, feststellen.

Nun mussten die bestimmten Noten einer gewählten Stimme zugeordnet werden. Dies wurde durch melodische Gruppierung verwirklicht. Zunächst wurden Inseln gebildet. Inseln sind Noten die sich Zeitlich vollständig überlappen. Wir gehen davon aus das jede Stimme nur eine Note gleichzeitig spielt, wodurch diese Noten nicht der gleichen Stimme zugeordnet werden können. Als Nächstes müssen die anderen Noten auf verschiedene Kombinationen getestet werden. Desto kleiner die Frequenz sprünge je Note sind, desto wahrscheinlicher gehören sie einer Stimme zu. Zudem werden Gruppierungen von Noten erstellt, welche am wahrscheinlichsten harmonisch nacheinander gespielt worden.

Zum Schluss ließ Moorer die gewonnenen Daten durch ein Programm laufen, um diese dann mithilfe eines Plotters in eine Notenschrift umzuwandeln.

2.1.3 Hidden Markov Models

Hidden Markov Modelle (HMM) sind statische Modelle, welche sich sehr gut zur Analyse von Musikstücken eignen. Sie wurden erstmal in den 1960er Jahren erfunden [1] und sind ein zentraler bestandteil früherer AMT-Systeme. HMMs beschreiben eine Abfolge von "versteckten Zuständen", welche im Kontext von AMT-Systemen Noten im Audiosignal darstellen. Durch indirekt beobachtbare Daten, wie zum Beispiel die Spektraldaten des Audiosignals, können diese Noten erschlossen werden.

HMMs bestehen aus:

- Zustände (States)
- Übergangswahrscheinlichkeiten (Transition Probabilities)
- Emissionswahrscheinlichkeiten (Emission Probabilities)
- Beobachtungen (Observations)

Nehmen wir das Beispiel eines Klaviers. Ein Klavier hat 88 Tasten, und somit mindestens 88 Zustände. Durch Akkorde können zudem mehr Zustände generiert werden. Die Übergangswahrscheinlichkeit stellt dar, wie wahrscheinliche es ist von einem Zustand zu einem bestimmten anderen Zustand zu wechseln. Zum Beispiel könnte es wahrscheinlicher sein, dass auf die Note C4 der Ton G3 folgt statt D1, da diese Tonfolge harmonischer und musikalisch plausibler klingt. Dies ist jedoch nur eine Annahme anhand von gesammelten Testdaten, weshalb es nicht als Begründung ausreicht. Deshalb kommt als zweite Instanz die Emissionswahrscheinlichkeit hinzu. Diese gibt an, wie Wahrscheinlich ein bestimmter Zustand in der momentanen Beobachtung ist. Die Beobachtung wird dabei zusammengesetzt aus Eigenschaften, wie Frequenzverteilung, Spektrogramm oder anderen Merkmalen die man aus anderen Modulen herleiten kann. Aus den gesammelten Daten kann nun mithilfe von zum Beispiel dem Viterbi-Algorithmus [39] die wahrscheinlichste Abfolge von Zuständen berechnet werden.

In gewisser Weise lassen sich HMMs auf KI-Modelle übertragen. Beide arbeiten mit versteckten Zuständen und berechnen anhand von Testdaten die Wahrscheinlichste kombination von Zuständen. Natürlich sind moderne KI-Modelle weitaus komplexer, doch HMMs spielten im Kontext automatischer Musiktranskriptionssysteme eine zentrale Rolle dabei, wie KI-Methoden in diesem Bereich später eingesetzt wurden. Sie beeinflussten insbesondere den Umgang mit zeitlichen Abfolgen und Unsicherheiten in der Tonerkennung, was bis heute relevante Konzepte in KI-basierten AMT-Systemen sind.

2.2 MIDI-Dateien

Notenschrift als Input für ein Computerprogramm, Synthesizer oder ähnliches ist unhandlich. Zunächst müsste man die gespielten Noten immer wieder zu Notenschrift konvertieren und danach diese auch noch in anderen Programmen analysieren, was sehr aufwändig werden würde. Eine bessere lösung dafür wäre eine Datenschreibweise, bei der alle wichtigen Informationen bestimmter Noten übersichtlich aufgeschrieben sind. MIDI-Dateien sind dafür perfekt geeignet.

MIDI ist ein Standartprotokoll zur kommunikation zwischen elektronischen Musikinstrumenten, Computern und anderen Geräten wie zum Beispiel Synthesizer. Dieses Protokoll wurde 1983 erstmals eingeführt und wurde schnell zu einem Standard in der digitalen Musikindustrie. [6] In MIDI-Dateien werden Daten von Tönen gelagert, welche zum Beispiel zuvor von einem elektronischen Instrument gespielt wurden oder durch AMTs erfasst wurden.

In MIDI-Dateien werden folgenden Daten gespeichert:

1. **MIDI Header-Chunk (MThd):** Enthält grundlegende Informationen zur Struktur der Datei:
    - Formattyp (0 = eine Spur, 1 = mehrere synchrone Spuren, 2 = unabhängige Spuren)

- Anzahl der folgenden MTrk-Blöcke (Tracks)
- Zeitauflösung (Ticks pro Viertelnote)

2. **MIDI Track-Chunks (MTrk):** Jede Spur enthält eine zeitlich sortierte Liste von MIDI-Events:
   - **MIDI-Events:**
     - Note Onset / Offset
     - Control Change (Lautstärke)
     - Program Change (gibt das spielende Instrument an)
     - Pitch Bend (verändert die Tonhöhe)
     - Aftertouch / Polyphonic Key Pressure (Druckstärke pro Taste)
   - **Meta-Events:**
     - Set Tempo (Tempo in Mikrosekunden pro Viertelnote)
     - Time Signature (Taktart zum Beispiel 4/4 oder 3/4)
     - Key Signature (Tonart zum Beispiel C-Dur oder A-Moll)
     - Track Name
     - Lyrics
     - Markers
     - End of Track (Ende einer Spur)
   - **System Exclusive Events (SysEx):**
     - Herstellerspezifische Daten wie Synthesizer-Presets oder Spezialbefehle
3. **Delta-Time:** Gibt die Zeit (in Ticks) an, die seit dem letzten Event vergangen ist:
   - Ermöglicht die genaue zeitliche Positionierung jedes MIDI-Events
   - Grundlage für das Timing und die rhythmische Struktur der Datei

Am wichtigsten sind dabei die MTrk-Blöcke, in denen die Daten der einzelnen Noten gespeichert werden. Dabei stellt ein Track die Eventliste einer ganzen Stimme dar, wie zum Beispiel die Melodiestimme, eine Violine, die Pedalsteuerung eines Klaviers oder Metadaten. Es fällt auf das diese vier Beispiele alle sehr unterschiedliche Aufgaben und bedeutungen haben. Das liegt daran, dass in MIDI-Dateien eher zusammenhängende Funktionen gespeichert werden und nicht nur Musiknoten.

MIDI-Dateien kamen auch der Forschung für AMT-Systemen sehr gelegen, da man nun ein standardisiertes output Format für diese Programme besaß. Später werden diese zudem sehr Essenziell bei dem Training KI basierter AMT-Systeme. [40]

2.3 Entwickelung von Datensätzen

Während der Forschung an neuen AMT-Systemen wurden immer wieder neue Datensätze genutzt, um das AMT-System zu bespielen oder heutzutage auch zu trainieren. In der Transkription von Musikstücken sind diese sehr relevant, da man nicht so leicht an nutzvolle Datensätze generieren kann und mit dem weiteren Einsatz von KI-Modellen genau diese sehr wichtig, in größeren Mengen, sind. Die ersten Datensätze, welche genutzt worden waren selbst kreierte Datensätze, welche lediglich einige Eigenschaften der Noten beinhalten. In den AMT-Systemen von Moorer[33] oder auch Martin[32] wurde solch ein Ansatz verfolgt. Durch die einführung von MIDI-Dateien wurde ein standard entwickelt, welcher jetzt in dem großteil, der AMT-Systeme eingebaut werden sollte. Auf der Grundlage von MIDI-Dateien wurden auch spezialisierte Datensätze erstellte. Zwei der größten und wichtigsten

sind MAPS (MIDI Aligned Piano Sounds) und MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization). Dabei ist MAPS ein Datensatz aus künstlich generierten Audiospuren, während MAESTRO ein ungefähr 200 Stunden langer Datensatz von realen isolierten Studioaufnahmen ist. Vor allem bei dem training neuer KI-Modellen sind solche großen Datensätze sehr hilfreich. Ein weiterer wichtiger Datensatz ist ADTOF (Annotated Drum Transcription Onset Features). Dieser besteht ausschließlich aus unharmonischen Instrumenten. In Moorers und anderen AMT-Systemen ist die Erkennung von unharmonischen Instrumenten ein großes Problem. KI-Modelle können mithilfe von dem ADTOF Datensatz sich speziell auf diese Instrumente vorbereiten. Es gibt auch Methoden schnell seine eigenen Datensätze zu generieren. Eine der neusten Methoden nutzt HMM- und Viterbi-basierte Alignment-Verfahren, um synthetische Audiosignale zu kreieren, damit man mehr Daten hat um sein KI-Modell zu trainieren. [21]

2.4 Polyphone AMT-Systeme und neue Ansätze

2.4.1 Blackboard-System

Moorer stellte, mit seinem AMT-System, viele grundlegende Bausteine für dieses Forschungsgebiet. Jedoch gab es noch viele offene Probleme, die noch überwunden werden mussten. Eines der ausschlaggebendsten Probleme war die polyphonie von Musikstücken. [32] Zuvor wurden höchstens zwei verschiedene Stimmen gleichzeitig zur Musik transkription verwendet. Ein großer Teil von Musikstücken verwendet jedoch mehr Stimmen. Um diese polyphonen Musikstücke zu transkribieren, baute Martin einen neuen Ansatz eines AMT-Systems mit innovativen Modulen und Ansätzen.

Nachdem das Input Audiosignal in ein Correlogram verarbeitet wurde, spaltet sich Martins AMT-System auf, in einen Analysepfad und einen Rhythmuspfad. Dabei konzentriert sich der Analysepfad auf die Zusammenhänge der verschiedenen Noten, während der Rhythmuspfad sich mehr mit der Lautstärke und den Notenanschlägen beschäftigt. Am Ende werden diese Kenntnisse zusammengeführt. Der Output besteht aus dem Onset, der Dauer und der Frequenz jeder gespielten Note. Martin gibt nicht explizit zurück, welcher Stimme jede Note zugeordnet ist. Durch das Blackboard-System, und vor allem nach sequentieller Analyse der Intervalle, kann man jedoch die erhaltenen Noten später bestimmten Stimmen nachträglich korrekt zuordnen.

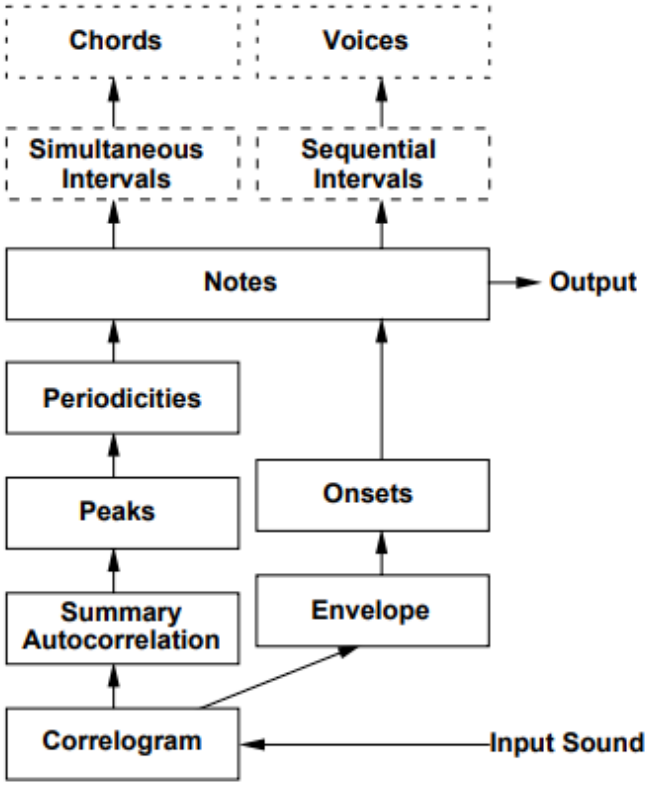


Abbildung 2: Systemstruktur des AMT-Ansatzes von Keith D. Martin (1996), basierend auf [32].

Martin nutzte ein Blackboard-System zur Notenerkennung und Hypothesenbildung. Unter anderem kann man dadurch Noten bestimmten Stimmen zuordnen. Ein Blackboard-System ist ein kollaboratives Problemlösungsmodell, welches aus verschiedenen Modulen besteht, die alle auf einem gemeinsamen Datenraum Hypothesen aufstellen können. Um diese Vermutungen aufstellen zu können müssen jedoch erst die dafür nötigen Daten erarbeitet werden. Dies erreicht Martin mithilfe von Sinuskurvenanalyse. Zur Sinuskurvenanalyse, in Martins Aufbau, gehören die Bestandteile Correlogram, Envelope, Summary Autocorrelation und Peaks. Durch diese können Eigenschaften der Noten ermittelt werden, auf denen aufbauend später Hypothesen entstehen. Die Module des Blackboard-Systems sind dementsprechend Onsets, Periodicities und Simultaneous, Sequential Intervals. Diese Module geben sich gegenseitig mehr Informationen über die gespielten Noten, sodass später gegebene Stimmen unterschieden werden können.

Der Analysepfad beginnt bei dem Correlogram. Hier wird der Input, ein reales Audiosignal, zunächst verarbeitet und dann in einem Correlogram dargestellt. Dieses Visualisiert die Korrelation der gegebenen Noten in Abhängigkeit der Zeit, Frequenz und Lag. Lag soll in diesem Fall darstellen, wann sich ein bestimmtes Signal wiederholt. Dabei ist die Wiederholung ein Signal, welches dem Ursprungssignal maximal selbst ähnelt. Heißt, wenn ich einen Ton mit der Frequenz x spiele und dieser sich nach zum Beispiel 10ms wiederholt wird das rechnerisch durch den Lag dargestellt. Als Nächstes wird mithilfe der Summary Autocorrelation das Correlogram komprimiert und die Datenstruktur normalisiert. Dadurch entsteht eine stabile Grundfrequenz, sodass die weiteren schritte effizient das Correlogram untersuchen können. Nun werden basierend auf der normierten Struktur die Peaks gesucht. Diese Peaks deuten darauf hin, wann periodische Komponenten auftreten. Dabei geben sie nicht den Onset der Noten zurück, sondern nur die generelle Frequenz zu einer bestimmten Zeit. Darauf werden mehrere Peaks, die regelmäßig wiederkehren, gruppiert. So kann man Muster in der gespielten Musik erkennen und kurzzeitige Störfaktoren ausschließen.

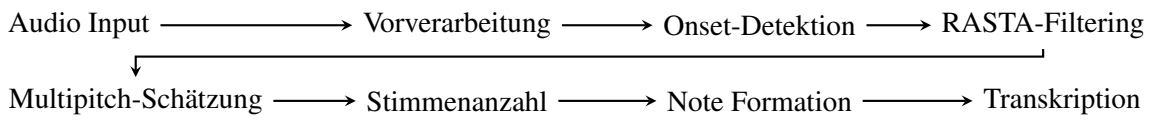
Im Rhythmuspfad wird wiederum die zeitliche Analyse der Noten durchgeführt. Zunächst wird mit dem Envelope die Lautstärke des Signals zu jedem Zeitpunkt festgelegt. Dies dient als Grundlage, um anschließend die Onsets der Noten zu bestimmen. Jeder Onset wird sehr präzise im Envelope

dargestellt durch einen plötzlichen Anstieg der Lautstärke.

Nachdem die beiden Pfade durchgelaufen sind, hat man schon Noten, welche man grundsätzlich in Notenschrift transkribieren kann. Jedoch sind diese Noten musikalisch noch nicht verstanden worden. In der oben gegebenen Abbildung gibt es noch die Bestandteile Simultaneous und Sequential Intervals. In dem Modul Simultaneous Intervals wird ermittelt, welche Töne gleichzeitig erklingen. Diese können, wenn sie harmonisch zueinander sind, zu einem Chord gebunden werden. Sequential Intervals analysieren dahingegen, welche Töne nacheinander gespielt werden und möglicherweise eine Melodie bilden könnten. Dies erfolgt durch Tonhöhenverläufe, Pausen und Frequenzunterschiede. Dadurch können Melodien, Basslinien oder Begleitungen voneinander getrennt werden. Zum Schluss werden dadurch die verschiedenen Stimmen, in Martins Modell, aufgetrennt.

2.4.2 RASTA-basiertes System

Ende 1997 publizierte Kalpuri seine Masterarbeit "Automatic transcription of Music". [25] In seiner Arbeit benutzt er ein neues RASTA-basiertes Verfahren, zur Unterdrückung nicht harmonischer Signalkomponenten. Somit konnten auch Musikstücke mit nicht harmonischen Instrumenten, wie Trommeln, transkribiert werden. Zudem führte er ein Modul ein, zur Schätzung der Anzahl an gleichzeitigen Stimmen, die in dem Musikstück vorkommen. Kalpuris AMT-System konzentriert sich, gegenüber Martins AMT-System, viel mehr auf die robustheit gegenüber echten polyphonen Audioaufnahmen mit rauschen, Störgeräuschen und nicht harmonischen Instrumenten. Die Struktur des Systems ist linear aufgebaut und sehr präzise in dem Aufgabenfeld, für die es entwickelt wurde.



Kalpuris System ist sequentiell aufgebaut und besteht aus sechs Modulen. Als Input wird eine reale Audioaufnahme genutzt, welche in ein Monosignal umgewandelt wird. Dieses Monosignal wird mithilfe von STFT transformiert, sodass durch das entstandene Spektrogramm Zeit, Frequenz und Amplitude der gegebenen Noten ausgelesen werden kann. Anhand dieser Daten werden als Nächstes die Onsets der Noten bestimmt. Kalpuri nutzt, zur Onset erkennung, ein Schema von Eric D. Scheirer. [35] Bei diesem wird das Audiosignal in Frequenzbänder aufgeteilt und auf jedem Band die zeitliche Änderung der Energie analysiert. Daraufhin werden die resultierenden Energie-Deviate der verschiedenen Bänder summiert. Die nun entstehenden Peaks werden als Onsets der Noten interpretiert. Als Nächstes werden unharmonische Instrumente und transiente Geräusche, wie zum Beispiel Rauschen, entfernt. Dies erfolgt mithilfe der RASTA-Filterung, welches auf das spektrale Signal, vom gegeben Spektrogramm, wirkt. Mehr zu der RASTA-Filterung werde ich in einem folgenden Absatz, im Detail, erläutern. Um mehrere Töne, die gleichzeitig im Signal erklingen, zu erkennen wird harmonic matching genutzt. Die Multipitch-Schätzung funktioniert so, dass zu einer bestimmten Zeit immer die dominanteste Tonhöhe, basierend auf der Energieverteilung im Signal, gesucht wird. Sobald man diese ermittelt hat, wird sie vom Signal subtrahiert und dieses Verfahren wird erneut eingesetzt, bis man jeden signifikanten Pitch untersucht hat. Jetzt werden die Stimmenanzahlen geschätzt. Dieses Modul werde ich ebenfalls ausführlich in einem folgenden Absatz detailliert wiedergeben. In dem Modul Note-Formation werden nun die gegebenen Daten zusammengeführt und in eine MIDI-Datei zusammengefasst. Diese MIDI-Datei wäre jetzt bereit in Notenschrift transkribiert zu werden.

Die RASTA-Filterung ist eine der neuen Module von Klapuri. Sie bewirkt, das nicht-harmonische Störsignale, wie Rauschen, unharmonische Instrumente und vom Musikstück unabhängige Geräusche ausgefiltert werden. Dadurch werden die harmonischen Komponenten des Stückes mehr herausgehoben, wodurch folgende Module einfacher weitere Eigenschaften den gegebenen Noten zuordnen können. Zudem stärkt dies die Robustheit der Multipitch-Schätzung, in der mehrere Töne zur gleichen Zeit im Audiosignal erkannt werden müssen. Dieses Verhalten wird erzielt, indem ein Filter gesetzt wird, der schaut, wie Laut jede Frequenz zu jedem Moment ist. Nach der Berechnung gibt der Filter

eine Lautstärke-Kurve zurück. Alle Frequenzanteile, die entweder zu kurzzeitig sind, wie etwa Claps oder Hi-Hats, oder zu langanhaltend, wie dauerhaftes Rauschen, werden durch einen Bandpassfilter aus dem Audiosignal herausgefiltert. Dadurch bleiben vor allem die zeitlich stabilen, musikalisch relevanten Komponenten erhalten.

Auch die Stimmanzahl erkennung ist, von Klapuri, ein neu eingefügtes Modul. Vor allem bei der Multipitch-Schätzung ist dieses sehr wichtig, da das System dadurch ein Bild davon bekommt, wie viele Töne gleichzeitig erklingen können, zu einer bestimmten Zeit. Die Multipitch-Schätzung zieht den spektralen Abdruck eines Tons von dem Audiosignal ab, wenn dieser erkannt wurde. Jedoch kann diese nicht einschätzen, wann genau sie aufhören soll die Töne zu erkennen. Da hilft die Stimmanzahlschätzung. Diese analysiert das neue Audiosignal nach jeder Multipitch-Schätzung und gibt aus, ob in der spektralen Energie noch Noten zu erkennen sind. Dies ordnet die Stimmanzahlschätzung durch drei Eigenschaften ein. Gibt es in der spektralen Energie noch typische Muster von harmonischen Klängen? Wie viele Stimmen wurden schon erkannt? Tragen die weiteren extrahierten Stimmen noch groß etwas zur Erklärung des gesamten Musikstückes bei? Dadurch extrahiert die Multipitch-Schätzung nur die nötigen Noten und unnötig Störfaktoren werden ausgelassen. Zudem kann man eine ungefähre Einschätzung bekommen, wie viele Stimmen in dem jeweiligen Musikstück erklingen.

2.5 Einbindung von Künstlicher Intelligenz

Für eine lange Zeit basierten AMT-Systeme größtenteils auf signal verarbeitenden Algorithmen, wie zum Beispiel HMMs. Doch auch diese Systeme waren nicht verschont durch den stetigen Anstieg von KI basierten Lösungswegen. 2010 bis 2012 wurden die ersten Ansätze von AMT-Systemen erstellt, welche maschinelles Lernen nutzten. [8] Oft wurden KI-Modelle nur für bestimmte Teilprobleme genutzt wie Onset Detection, pitch estimation oder instrument classification. Dieser Trend zog sich durch, wodurch heutzutage so gut wie jedes AMT-System eine Art von KI-Modell nutzt.

Einige AMT-Systeme nutzen auch mehrere KI-Modelle gleichzeitig. Solch ein AMT-System wird auch einer 2016 veröffentlichen Arbeit beschrieben. [37] Es ist eins der ersten KI basierenden Ent-to-End AMT-Systeme, welches polyphone Musikstücke transkribieren kann. Der KI-Ablauf des AMT-Systems ist folgendermaßen aufgebaut. Ein CNN bekommt als Input ein Log-Mel-Spektrogramm des Audiosignals bereitgestellt und entzieht diesem bestimmte Merkmale der Noten. Diese Merkmale werden einem RNN weitergegeben, welches dadurch die zeitlichen Abhängigkeiten herausfiltert. Als Nächstes wird durch Frame-wise Multi-Label Classification vorhergesagt, welche Noten zu welcher Zeit aktiv sind. Somit bekommt man als Output eine Binärmatrix, wo der Pitch mit Abhängigkeit der Zeit angezeigt wird. Das System wird mit dem MAPS Datensatz durch Supervised Learning trainiert. Diese Arbeit stellt den drastischen übergang von heuristischen, regelbasierten AMT-Systemen zu selbstlernenden AMT-Systemen dar. Heutzutage sind CNNs und RNNs, oder ähnliche KI alternativen, kaum wegzudenken aus der Struktur von AMT-Systemen.

Viele intelligente und erfindungsreiche Personen haben sich über die Jahre mit AMT-Systemen beschäftigt. Dadurch konnten nachfolgende Arbeiten neue Module und Ansätze für sich nutzen, um deren Entwicklung weiter voranzutreiben. Auch heutzutage ist dies immer noch der Fall, wodurch es stets neue AMT-Systeme mit neuen Ansätzen gibt. Diese neuen, auf KI basierenden Systeme, haben alle jeweils ihre eigenen Stärken und schwächen. Auch wenn die Forschung von AMT-Systemen in den letzten Jahren viele Fortschritte erfuhr, gibt es trotz dessen noch große Baustellen und ungelöste Aufgaben. Im folgenden Abschnitt werde ich weiter auf den momentanen Stand von AMT-Systemen eingehen und anhand einiger Beispiele verschiedene Strukturen darstellen. Zudem gehe ich auf momentan relevante Hindernisse in der automatischen Musiktranskription ein.

3 Hindernisse Moderner AMT-Systeme

Trotz der Einführung von KI-Modulen gibt es noch viele offene Probleme, die noch nicht gelöst werden konnten oder noch nicht perfekt gelöst sind. Auch wenn durch die nutzung von KIs viele

Fehler geringer oder sogar komplett behoben werden konnten, agieren diese völlig anders als normale Algorithmen. Dies führte, zum Teil, auch zu neuen Problemen, welche davor nicht mal bekannt waren.

3.1 Onset Detection

Die On-/Offset erkennung von Noten wurde schon ausgiebig in vielen Arbeiten behandelt. Trotzdem ist diese noch nicht komplett akkurat. Das liegt daran, dass man On-/Offsets an Lautstärke sprüngen und spektralen Änderungen erkennt. Bei polyphonen Musikstücken mit vielen verschiedenen Noten kann man jedoch schwieriger diese Unterschiede erkennen. Lautstärke sprünge werden ungenauer, da viele andere Noten während des Onsets einer bestimmten Note spielen. Mit spektralen Änderungen sind hier Veränderungen der Energieverteilung gemeint. Einer der ausschlaggebendsten Anteile ist die Spektrale Fluktuation. Diese stellt den plötzlichen Anstieg von Energie in bestimmten Frequenzbändern dar. Wenn nun ein Ton auf einem Klavier gespielt wird kann somit der Onset sehr gut ermittelt werden. Bei Instrument wie einer Geige kann dies jedoch zu problemen führen. Hier können Noten gebunden gespielt werden, was zu einem unterschied in der Frequenz, jedoch nicht in dem Energie level führt. Zudem kann eine Note, durch Crescendo, zunächst leise gespielt werden und mit der Zeit an Lautstärke zunehmen. Dieser Onset hat keinen Energie-Peak und somit erkennt das System hier auch nur schwierig eine scharfe Kante. So welche Töne, welche keinen starken Einschlag haben, werden als nicht-perkussiv bezeichnet. Ein weiteres Problem der Onset erkennung sind nach wirkende Geräusche oder stör Geräusche. Bleiben wir bei dem Beispiel einer Geige, so können einige Töne, sobald man aufhört andere Töne zu spielen, nachklingen. Dadurch könnten Töne von anderen Instrumenten verdeckt werden, wodurch das System den Onset nicht erkennt. Ähnlich kann dies auch der Fall sein, wenn im Audiosignal ein starkes Rauschen besteht.

3.2 Polyphonie und Notenzuordnung

Durch die nutzung von polyphonen Musikstücken sind, wie schon bei der Onset Erkennung, viele Probleme schwerer und deutlicher geworden. Ein weiteres Problem, das gezielt von diesen Anwendungsfällen abhängt, ist die Verwechselung von Noten im Audiosignal. Einige Noten haben sehr ähnliche Frequenzen. Wenn diese Noten gleichzeitig gespielt werden, ist es schwieriger für das System, die korrekten Noten herauszuhören. Neuronale Netzwerke helfen hierbei deutlich, da diese nicht nur das Spektrum zur analyse einbeziehen, sondern auch auf einer zeitlichen und harmonischen Ebene den Kontext besser zuordnen können und somit die wahrscheinlichsten nächsten Noten zurückgeben können. Trotz dessen scheitert auch KI an der einordnung der richtigen Noten, wenn die gespielten Noten eine sehr ähnliche Frequenz besitzen oder die Akkorde zu unterschiedlich zu den Trainingsdaten sind. In der folgenden Arbeit werden diese Probleme nochmals deutlicher aufgegriffen. [31]

3.3 Instrument spezifische Probleme

Oft ist die Qualität der Transkription auch abhängig von den vertretenen Instrumenten in dem Audiosignal. Ethnische Instrumente zum Beispiel sind Instrumente die einer bestimmten Kultur angehören und in unserer westlichen Kultur weniger vertreten sind. Dadurch gibt es auch weniger Datensätze, welche diese Instrumente beinhalten. KIs brauchen Trainingsdaten und ohne diese können sie bestimmte Instrumente nicht richtig zuordnen. Die meisten Datensätze zur Musiktranskription bestehen hauptsächlich aus Klavier und Geigen Noten. Instrumente wie Flöten oder Orgeln können von diesen Noten gut abgeleitet werden, da sich deren Struktur deutlich ähnelt. Eine weitere Gruppe von Instrumenten die Probleme bei der Transkription bereitet sind elektronische Instrumente. Wenn man zum Beispiel eine E-Gitarre spielt, kann diese Effekte nutzen, welche nicht üblich in klassischen Datensätzen von Klavieren vorkommen. Somit erkennt die KI diese Töne nicht korrekt an. Das letzte Instrument welches schwierigkeiten bringt, ist der Gesang. Jede Stimme ist einzigartig und vor allem nicht statische. Wenn man einen Ton auf dem Klavier spielt, besitzt dieser, wenn das Klavier richtig gestimmt ist, immer die gleiche Frequenz. Ein Mensch kann jedoch nicht jeden Ton immer komplett perfekt spielen, wodurch eine große varianz an Tönen entsteht. Zudem verläuft der Klang einer Stimme von einer Note zur nächsten. Es gibt nicht immer starke Peaks zur Onset Erkennung.

Gesang ist auch, wie die vorher genannten Instrumentgruppen, nicht sonderlich vertreten in größeren Datensätzen. Die folgenden Paper konzentrieren ihren Fokus speziell auf das Thema des Gesangs in der Musiktranskription. [14, 13] Somit wurde eine Onset Erkennungsgenauigkeit von ungefähr 80% festgestellt, für Gesang.

3.4 Notendauer

Neben dem Onset einer Note muss man auch erkennen, wie lange eine bestimmte Note gespielt wird. Das kann schwierig sein, da man den Nachhall einer Note von der wirklich gespielten Zeit der Note trennen muss. Es gibt jedoch bei einigen Instrumenten, wie zum Beispiel dem Klavier, ein spezielles Problem. Wenn man während des Spielens einer Note das Pedal drückt, gibt es keinen klaren Punkt, an dem man den Übergang zwischen der Note und dem Nachhallen eindeutig erkennen kann. Die Lautstärke sinkt dabei nicht abrupt, sondern nur langsam ab. Natürlich ist dieses Problem in polyphonen Musikstücken nochmal deutlich stärker, da sich dort die verschiedenen Nachhallphasen überlappen. Dies ist eins der grundlegendsten Probleme, zusammen mit der Onset Erkennung. [19]

3.5 Reale Audioaufnahmen

Ein perfektes AMT-System sollte sogar auf realen Audioaufnahmen 100% Genauigkeit besitzen. In der Realität klappt das aber nicht so gut, da Live-Audioaufnahmen mehr Hintergrundrauschen besitzt. Dies war schon vor der Einführung von KI ein Problem und wurde durch die Einbindung von KI-Modulen nicht sonderlich viel verbessert. Das liegt daran, dass KIs mit isolierten Studioaufnahmen, wie es im MAESTRO Datensatz der Fall ist, trainiert werden. Natürlich gibts auch Datensätze mit realen Audioaufnahmen, jedoch bräuchte man dafür viel mehr Trainingsdaten und vor allem auch viele unterschiedliche Hintergrundgeräusche, damit die KI auf alles trainiert wird. Um dem entgegenzuwirken, kann man Studio-Datensätze wie MAESTRO durch Data-Augmentation variieren, wodurch realistischere Audioaufnahmen entstehen. So wurde es auch in folgender Arbeit, für die Trainingsdaten, eingesetzt. [26] Es wird Timber, Reverb, Noise variierte und die Qualität des Aufnahmegerätes, zu der eines Smartphones, veränderte. So bekommt man viele neue Datensätze, die passend auf die gegebenen Anwendungsfall ausgelegt sind.

3.6 Frame-basierte vs Event-basierte AMT-Systeme

Die meisten AMT-Systeme sind Frame-basiert und nicht Event-basiert. Frame-basiert bedeutet, das die Analyse des Audiosignals in Zeitfenster, ungefähr 20ms, aufgeteilt wird. Jedes Zeitfenster wird somit der momentane Stand das Audiosignal von den verschiedenen Modulen analysiert. Falls nun zum Beispiel ein Onset einer Note genau zwischen zwei Zeitfenstern liegt, wird dieser verschoben oder verzerrt, sodass dieser mit den gegebenen Zeitfenstern übereinstimmt. Event-basierte AMT-Systeme hingegen fragen immer ab, was das nächste musikalische Ereignis im Audiosignal ist und reagieren dementsprechend. Dadurch entspricht der Output viel mehr dem Input, in relation zu der zeitlichen Abfolge. Im Ergebnis sind Event-basierte AMT-Systeme somit Frame-basierten AMT-Systemen überlegen. Jedoch ist in der Realität trotzdem fast jedes AMT-System Frame-basiert. Das liegt daran, dass Event-basierte AMT-Systeme eine sehr neue Entwickelung sind und zudem auch schwerer zu implementieren ist. Frame-basierte Methoden wurden ungefähr im Jahre 2000 entwickelt und haben sich seitdem in zahlreichen Arbeiten durchgesetzt. [32, 25] Dahingegen sind Event-basierte Methoden erst ungefähr 15 Jahre später entwickelt worden. [30] Deshalb wurden Frame-basierte Methoden lange als der Standard angesehen. Zudem ist die Umsetzung von Event-basierten AMT-Systemen schwerer als Frame-basierte. Das System muss korrekt jedes Event einschätzen un zuordnen, ist es ein Onset ein Offset oder doch nur Rauschen? Wenn es nun einen Fehler macht, wird sich das, im gegensatz zu Frame-basierten, auf das ganze weitere Audiosignal auswirken. Auch das Training kann nicht parallel verlaufen, sondern muss token weise abgearbeitet werden. Diese Gründe, und noch einige andere, machen Event-basierte AMT-Systeme zu einer viel größeren Herausforderung. Jedoch ist der übergang von Frame zu Event mit der Zeit irgendwann nötig, da man somit den Inhalt der Musikstücke viel präziser wiedergeben kann. Die Forschung in dem Event-basierten

Methoden steigt auch in den letzten Jahren immer mehr an.

3.7 Blackbox von KI-Modellen

Das letzte Problem betrifft KI im generellen. Ein neuronales Netz arbeitet mit sehr hochdimensionalen Räumen, die für uns Menschen nicht begreifbar sind. Selbst wenn wir uns die Millionen und mehr Gewichtungen anschauen würden in diesen keinen Zusammenhang feststellen. In der Musiktranskription ist dies auch ein Problem, da wir somit nicht erfassen können, welche Einstellungen unsere KI genau braucht damit sie alle Musikstücke perfekt transkribieren kann. Jedoch gibt es auch Methoden, welche die Blackbox ein wenig umgehen. Im Forschungsgebiet der Explainable AI gibt es einige Methoden, welche auch in KI-basierten AMT-Systemen, verwendet werden. Unter diesen fallen Saliency Maps, Feature-Visualisierung und Attention-Mapping. Concept Activation Vectors, Layer-wise Relevance Propagation, Surrogat-Modelle sind andere Explainable AI Methoden, die zur nutzung in AMT-Systemen diskutiert werden, aber noch nicht richtig integriert sind. Dies liegt an mehreren Gründen. Musikalische Konzepte wie Akkorde oder Onsets sind nicht direkt lablebar, wie zum beispiel Katzen oder Hunde, da diese stark abhängig von spektralen und zeitlichen Mustern sind. AMT-Systeme sind zeitlich-sequenziell und besitzen pro Zeitfenster mehrere Outputs. Viele lokale Änderungen auf ein bestimmtes Zeitfenster, wie den dB-Wert ändern, haben einen globalen Einfluss. Methoden, welche in AMT-Systemen angewendet werden, haben dahingegen Eigenschaften, die durch die Struktur von AMT-Systemen profitieren. Saliency Maps stellen dar, wie stark jedes Zeitfenster im Spektrogramm, beeinflusst hat, das zum Beispiel ein bestimmter Ton erkannt wurde. Dies basiert auf dem Gradienten und ist besonders hilfreich bei CNN-basierten Modellen. Feature-Visualisierung beobachtet konkret die einzelnen Neuronen jeder Ebene im neuronalen Netzwerk. Es wird geschaut, welche neuronen sehr stark bei bestimmten Inputs reagieren. So kann man erkennen, welche Teile des Netzes für welche Aufgaben verantwortlich sind und ob vielleicht bestimmte stellen des Netzes gar nicht genutzt werden. Attention-Mapping nimmt den gegebenen Input und gewichtet diesen, je nachdem welche Teile davon wichtig für eine bestimmte Vorhersage sind. So erkennt man, welche Zeitfenster den meisten Einfluss auf zum Beispiel einen bestimmte Onset hatten. Vor allem bei polyphonen Musikstücken kann man relevante zeitliche Zusammenhänge erkennen. Ein gutes Beispiel von Attention-Mapping findet man in folgender Arbeit. [4]

4 KI-Module in AMT-Systemen

KI-Systeme haben, in den letzten Jahren stark an beliebtheit gewonnen. AMT-Systeme bilden da keine Ausnahmen. Vor allem durch die integration von CNNs und RNNs konnten AMT-Systeme viele Prozesse deutlich verbessern und neue Errungenschaften in dem Forschungsgebiet erzielen. Es gibt jedoch auch weitere wichtige KI-Module die in AMT-Systemen häufig genutzt werden oder zur integration in Planung stehen. In diesem Kapitel werde ich auf genau diese KI-Module stärker eingehen und deren Aufgaben in der automatischen Musik transkription weiter erläutern.

4.1 Convolutional Neural Networks

CNNs sind neuronale Netze, welcher besonders gut räumlich strukturierte Daten analysieren können. Deshalb werden diese vor allem in der analyse von Bildern genutzt. Sie können zum Beispiel erkennen, was auf einem Bild genau passiert oder welche Objekte in einem Bild zu erkennen sind. Auch KIs wie ChatGPT nutzen eine verbesserte Form von CNNs, um Bilder zu analysieren. Im Fall der Musiktranskription haben wir als Input Bild das Spektrogramm. Spektrogramme können ähnliche wie zweidimensionale Bilder gehandhabt werden, da auf diesen auch alle wichtigen Daten des Inputs Audiosignals zu finden sind. CNNs bestehen aus mehreren verschiedenen Layern. Einfache CNN Modelle bestehen nur aus 2 bis 5 Layern, wobei komplexere CNNs aus über tausende Layern bestehen können. In AMT-Systemen haben die meisten CNNs etwa drei bis zehn Layer. Diese Layer sind Convolutional Layer (Faltungsschicht), Activation Layer (ReLU), Pooling Layer, Batch Normalization, Dropout Layer und Upsampling. Mit jedem Layer kann ein CNN immer abstraktere Merkmale erkennen. Außer dem Convolutional Layer und Activation Layer sind die anderen Layer jedoch

nicht unbedingt notwendig. Eine Arbeit, welche die stärke, von CNN-Modellen in AMT-Systemen, sehr gut darstellt, heißt "Onsets and frames". [16] In dieser Arbeit werden direkt zwei verschieden spezialisierte Teilnetzwerke für Onsets und Sustain der Noten. Mit diesem System wird die Entwickelung des Forschungsgebietes illustriert. Insbesondere für polyphone Klaviertranskription ist dieses AMT-System ausgezeichnet. Im Folgenden werde ich die verschiedenen Layer eines CNNs, in einem AMT-System, erläutern.

4.1.1 Layer eines CNNs

In dem Convolution Layer werden Filter verwendet. Filter sind 2D-Matrizen, die aus trainierbaren Gewichten bestehen. Ein Filter deckt jeweils einen 3x3 Pixel Eingabebereich des Inputbildes ab. Jeden Filter, den man auf das Bild anwendet, wird über das gesamte Bild gezogen und analysieren dadurch erstmal jeden Eingabebereich einzeln. Ein Skalarprodukt aus Filter und Eingabebereich beschreibt dann einen Aktivierungswert. Aus allen Aktivierungswerten eines Filters entsteht eine Feature Map. Wenn man Aktivierungswerte miteinander vergleicht, können somit Patterns und Eigenschaften erkannt werden. In der Musiktranskription filtert man somit Onsets, Sustains oder harmonische Verläufe heraus. Zum Beispiel Onsets werden erkannt, wenn es eine plötzliche Energieänderung gibt. Am Ende bekommt man einen 3D-Tensor raus, welcher alle Feature Maps beinhaltet.

Die Batch Normalization sorgt dafür, das die Aktivierungswerte normalisiert werden. Jede Feature Map wird dabei einzeln normalisiert. Dadurch wird Rechenleistung eingespart. Zudem kann man im Training durch Mini-Batches mehrere Spektrogramme gleichzeitig durch die CNN Struktur leiten. Dadurch wird das Training schneller und man kommt früher an Ergebnisse.

Es kann passieren das die Summe eines Convolution Layers negativ ist. Dies kann passieren, wenn stärker gewichtete Filter einen negativen Aktivierungswert herausgeben. Negative Werte können zu Informationsverlust, von Eigenschaften des Musikstückes, führen. Um das zu vermeiden werden im Activation Layer, meistens mit der ReLU Funktion, alle negativen Aktivierungswerte auf 0 gesetzt. So kann das Netz nichtlineare Beziehungen modellieren.

Als Nächstes wird mit dem Pooling Layer der Rechenaufwand verringert. Dieser nimmt jede Feature Map einzeln und reduziert deren Matrix zu einer kleineren, meistens 2x2, Matrix. Das erfolgt, indem sich der Pooling Layer zunächst eine gesamte Feature Map nimmt und diese dann in kleinere Blöcke aufteilt. Es gibt entweder Max-Pooling oder Average-Pooling. Je nachdem welche Methode man wählt, wird immer der höchste Wert oder der durchschnittliche Wert extrahiert. Der extrahierte Wert von jedem Block wird nun in die reduzierte Feature Map zurückgeführt. Dadurch reduziert man nicht nur Rechenaufwand, sondern vermeidet auch überanpassung. Wenn das System jeden kleinsten Wert berücksichtigt passt es sich zu sehr an den trainingsdaten an und kann womöglich andere Daten nicht mehr richtig analysieren.

Der Dropout Layer ist, im gegensatz zu den anderen Layern, nur im Training relevant. Er schaltet zufällig bestimmte Neuronen aus, sodass sich Neuronen nicht ausschließlich auf bestimmte andere Neuronen verlassen können. Somit wird das gesamte neuronale Netzwerk robuster und vielseitiger.

Upsampling ist das Gegenteil von dem Pooling Layer. Anstatt die Feature Maps zu reduzieren, werden diese wieder hochskaliert. Dadurch kann man bestimmte Features wieder zeitlich präziser bestimmen, da die Zeitfenster wieder genauer zum originellen Audiosignal sind. Meist wird dieser Layer jedoch weggelassen, da er meistens nicht sehr relevant für AMT-Systeme ist.

4.1.2 Geschichte von CNNs

CNNs finden Ihren Ursprung im Jahre 1979. Die erste richtige Architektur für CNNs wurde, unter dem Namen "Neocognitron", veröffentlicht. [9] Dies sollte als Vorreiter für spätere CNN Modelle gelten. Es dauerte weitere 10 Jahre bis das erste richtige Convolutional Neural Network veröffentlicht wurde. [28] Dieses CNN Modell unterscheidet sich speziell in drei bestimmten Punkten zu dem Vorgänger Neocognitron. In diesem wurde Gradientenlernen durch Backpropagation integriert, wodurch das gesamte Netzwerk erstmals auf ein gemeinsames Ziel zu trainieren konnte. Neocognitron

besaß zudem, im Gegensatz zu LeCuns CNN, keine Gewichtsverteilung mithilfe von Filtern, wie es in heutigen CNN Modellen standard ist. Zudem hatte Fukushima keine praktische Anwendung. Er hatte die Idee und wie man diese umsetzt, doch durch damalige Verhältnisse war es für ihn schwierig diese umzusetzen. Für AMT-Systeme wurde CNN jedoch erst ungefähr im Jahre 2015 relevant. [37] In dieser Arbeit wurde erstmals polyphone Musikstücke mithilfe von CNNs, und weiteren KI-Modulen, transkribiert. Seitdem sind CNNs ein wichtiger bestandteil der modernen automatischen Musiktranskription.

Ein CNN gibt als letzte Ausgabe einen 3D-Tensor zurück, welcher aus den Zeitfenster (Frames / T), der Frequenzachsenlänge (F) und der Anzahl der Filter (C) besteht.

$$\text{CNN-Ausgabe} \in \mathbb{R}^{T \times F \times C}$$

Meist schließt sich nach einem CNN, in einem AMT-System, ein RNN als Nächstes an. Dieses kann jedoch nur 1D-Vektoren verarbeiten und nicht einen 3D-Tensor. Deshalb wandeln wir vor der übergabe diesen 3D-Tensor um. Jedes Zeitfenster aus dem 3D-Tensor wird einzeln zu einem 1D-Vektor umgewandelt. Dabei wird der Vektor eine Dimension von F × C erhalten. Diese Vektoren werden dann an das folgende RNN weitergeleitet.

4.2 Recurrent Neural Networks

RNNs sind neuronale Netze, welche entworfen wurden um Daten mit zeitlicher Struktur zu verarbeiten. Dabei ist die besonderheit von RNNs das diese ein Gedächtnis haben. Wenn nun in unserem Fall von AMT-Systemen eine bestimmte Tonabfolge gespielt wurde, kann sich das RNN diese merken und dementsprechend die fortlaufenden Ausgaben anpassen. Dies passiert dank den Hidden States. Dieser stellt das Gedächtnis des RNNs dar und ist einer der wichtigsten Faktoren in einem RNN. Mehr zu den Hidden States erläutere ich im übernächsten Paragraf. Dieses Prinzip ist in der Musiktranskription sehr hilfreich, da jede Note stark von den vorherigen gespielten Noten abhängt. Takt, Rhythmus, Harmonie und die Melodie eines Musikstückes sind alles gute Beispiele, warum diese sequenzielle Abfolge so passend in AMT-Systemen ist.

4.2.1 Grundstruktur eines RNNs

RNNs haben in der automatischen Musiktranskription mehrere Aufgabenfelder. Je nachdem wie man das RNN trainiert bewältigt dieses alle Aufgaben oder nur einen Teil. Diese Aufgaben bestehen aus Frame-Glättung (Temporal smoothing), Kontext-Modellierung (Temporal context modeling), Feature-Zusammenführung (Sequential integration of acoustic features) und Ausgabevorbereitung (time-distributed output classification). Diese Aufgaben werden auf jedes Zeitfenster, auf jeden 1D-Vektor des CNNs, angewendet. Doch bevor man sich auf diese einzelnen Aufgaben konzentrieren kann, muss man noch wissen, was Hidden States sind.

Hidden States sind das grundlegende Prinzip, warum RNNs funktioniere. Sie stellen praktisch das Gedächtnis des RNNs dar und helfen somit anderen Modulen Vorhersagen über bestimmte musikalische Eigenschaften zu treffen. Für jeden 1D-Vektor, den wir vom CNN geliefert bekommen, wird ein Hidden State erstellt. Diese werden sequenziell nacheinander, mit abhängigkeit des vorherigen Hidden States, definiert.

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$$

In Hidden States ist das Wissen der vorherigen Zeitfenster über zeitliche Muster, wie Sustain und Akkordstruktur. Sie alleine können jedoch noch keine eigene Entscheidung über bestimmte Noten fallen. Dafür müssen andere Module die Informationen der Hidden States später richtig verwerten. Jeder Hidden State hat eine Anzahl von Dimensionen, welche gleich der Anzahle der Neuronen in einem RNN ist.

4.2.2 Aufgaben eines RNNs

Bei der Frame-Glättung bekommt das RNN als Input die Aktivierungswerte des Zeitfensters. Es betrachtet diese mit Kontext zu den vorherigen Zeitfenstern, um falsche vorhersagen auszuschließen. Das CNN hat dem RNN schon Vorhersagen für bestimmte Noten gegeben. Diese könnten jedoch fehlerhaft. Zum Beispiel kann die Note C4 für den Frame 6 und 8 aktiv sein, aber bei dem Frame 7 hat das CNN diese Note nicht als aktiv angesehen. Es ist sehr unwahrscheinlich, das eine Note für nur einen Frame ausfällt. Solche Arten von Fehlern verarbeitet das RNN und glättet dementsprechend die vorher vom CNN gelieferten Daten. Dadurch ist der Ton C4 auch im Frame 7 aktiv.

Die Kontext-Modellierung ist etwas komplizierter als die Frame-Glättung. Als Input bekommt diese auch die 1D-Vektoren des CNNs. In der Kontext-Modellierung werden größere zeitliche Zusammenhänge betrachtet. So kann die Kontext-Modellierung, mithilfe des Hidden States, den Notenverlauf oder auch die länge einer Note vorhersehen. Je nach den Trainingsdaten ist es zum Beispiel üblicher das auf C4 ein D3 folgt, was durch die Kontext-Modellierung angepasst wird. Dies ist aber immer stark von der Musikrichtung und Musikern abhängig, welche in den Trainingsdaten vorhanden sind. Jazz und Pop oder Bach und Taylor Swift unterscheidet sich der style der Tonabfolge extremst. Zudem wird Onset, Sustain und Offset stabilisiert. Durch vorherige Beispiele weis die Kontext-Modellierung, wie lange eine bestimmte Note andauern wird und kann so das Offset der Note einschätzen. Als Output bekommt man Kontextabhängige Vektoren. Sie haben die gleiche Struktur wie die 1D-Vektoren vom CNN, sind aber entsprechend dem Kontext angepasst.

Die Feature-Zusammenführung ist der letzte wichtige interne Schritt eines RNNs. Bei diesem werden aus, lokalen alleinstehenden, Informationen eines Zeitfensters, konkrete musikalische Ereignisse. Dadurch schreibt das RNN Ereignisse, wie Akkorde, Noten, Onsets und vieles mehr, heraus. Dafür muss die Feature-Zusammenführung sich die einzelnen 1D-Vektoren als eine folge von Events anschauen. Dies passiert wiedermal über den Hidden State. Das wird vor allem wichtig, wenn man später eine MIDI-Datei ausgeben möchte, da in dieser auch die einzelnen musikalischen Ereignisse aufgeschrieben sind. Als Output bekommt man kontextreiche Vektoren. Diese Vektoren sind, durch die vorherigen Module angepasste und verbesserte, Hidden States.

Die Ausgabevorbereitung ist der letzte Schritt des RNNs, wodurch nun die gesammelten Daten zu echten musikalischen Ereignissen zusammengefügt werden. Als Input werden die, durch die Feature-Zusammenführung verbesserten, Hidden States genutzt. Zunächst werden diese durch ein Fully Connected Layer geschickt.

$$\mathbf{z}_t = W \cdot \mathbf{h}_t + \mathbf{b}$$

Ein Fully Connected Layer ist ein Klassifikator, welcher den Hidden States auf eine gewünschte Dimension bringt. Wenn man zum Beispiel Klaviertasten vorhersagen möchte werden alle Hidden States auf mit 88 Dimensionen ausgestattet. Bei MIDI-Dateien wären es 128 Dimensionen. Die Werte, welche wir aus dem Fully Connected Layer herausbekommen sind, jedoch noch nicht richtig interpretierbar. Um diese als konkrete und normalisierte Wahrscheinlichkeiten darstellen zu können, nutzen wir noch eine Aktivierungsfunktion. Zum Beispiel können wir mit der Sigmoid-Funktion als Aktivierungsfunktion, alle Werte normiert in einem Bereich zwischen 0 und 1 bringen. Sagen wir, wir wollen jetzt die gespielten Klaviertasten vorhersagen. Dann hat jeder Hidden State jetzt für alle Klaviertasten einen eigenen Wert mit einer Wahrscheinlichkeit, das diese Taste zu dem gewählten Moment gespielt wurde. Zu guter Letzt müssen wir noch einen Threshold bestimmten. In polyphonen Musikstücken können immer mehrere Noten gleichzeitig erklingen, weshalb man nicht einfach die Note mit der höchsten Wahrscheinlichkeit auswählen kann. Deshalb nutzt man einen Threshold, zum Beispiel bei 50%, welcher bestimmt wie viel Prozent eine Note braucht um als aktiv zu gelten. Durch Postprocessing können einige Eigenschaften wie Rauschen noch herausgefiltert werden. Postprocessing ist jedoch nicht relevant für den KI-Ablauf. Wenn man zufrieden mit dem Ergebnis ist, kann man dieses jetzt in das gewünschte Output-Format, meistens MIDI-Dateien, einfügen.

Heutzutage sind RNNs sind nur die grundlegende Struktur. Basierend auf dieser Struktur gibt es einige bessere Systeme, welche Aktiv in AMT-Systemen und anderen KI-Systemen genutzt werden.

Zwei dieser Systeme sind Long Short-Term Memory's (LSTM) und Bidirektionale RNNs (BiRNN). Auf diese werde ich in den folgenden Kapiteln weiter eingehen.

4.2.3 Long Short-Term Memory

LSTMs sind verbesserte RNNs. Diese kontrollieren durch Gates besser, welche Daten sie wirklich in den Hidden State speicher möchten. Dadurch kann man das neuronale Netz noch weiter an seine gewünschten Ansprüche anpassen. Ein normals RNN berechnet einen Hidden State mit folgender Formel:

$$h_t = \tanh(Wx_t + Uh_{t-1} + b)$$

Dabei werden alle Daten, egal ob sinnvoll oder nicht, miteinander kombiniert. Somit kann sich das RNN langfristig schwieriger Information merken. Wenn zum Beispiel im 2Ḣidden State ein Onset erkannt wurde, kann der 20Ḣidden State sich das schlechter merken, da ganz viele andere Informationen mitgeschrieben wurden. LSTMs lösen dieses Problem mit Forget, Input und Output Gates und dem Cell State. Der Cell State stellt das Langzeitgedächtnis des LSTMs dar. Er berechnet sich aus den 3 Gates. Das Forget Gate bestimmt, welche Informationen aus dem vorherigen Cell State gelöscht werden sollen. Das Input Gate bestimmt, welche neuen Inhalte aus dem neuen Zeitfenster aufgenommen werden. Dabei ist der Cell-candidate die Datenmenge, welche zum Speichern, durch das Input Gate, vorgeschlagen wird. Das Output Gate bestimmt, welcher Teil des Cell States zu dem neuen Hidden State hinzugefügt wird.

$$\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) && \text{(Forget Gate)} \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) && \text{(Input Gate)} \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) && \text{(Output Gate)} \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) && \text{(Cell-candidate)} \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && \text{(Cellstate)} \\
h_t &= o_t \odot \tanh(c_t) && \text{(Aktueller Hidden State)}
\end{aligned}$$

Dadurch kann sich ein LSTM die verschiedenen musikalischen Ereignisse, über das gesamte Audiosignal, besser im Zusammenhang merken.

4.2.4 Gated Recurrent Units

GRUs sind, neben LSTMs, eine weitere spezielle Art von RNNs. [5] Sie wurden erfunden, um bestimmte Probleme von RNNs zu lösen. Insbesondere das Problem des "Vanishing Gradients" sollten GRUs lösen. GRUs steuern mithilfe von Gates, welche Informationen gemerkt und welche vergessen werden sollen. Sie sind, im gegensatz zu LSTMs, eher für kleinere Aufgaben geeignet. Dafür sind sie weniger fehleranfällig und haben ein schnelleres Training.

Ein GRU besitzt zwei Gates. Sie haben, genau wie bei LSTMs, die Aufgabe das Gedächtnis des KI-Models zu verwalten. Das Reset Gate schaut sich den alten Hidden State an und entscheidet, wie viel von diesem Wissen in den neuen Hidden State mit einfließt. Das Update Gate hingegen sieht den aktuellen Hidden State und überlegt, welche Informationen davon in das Langzeitgedächtnis mit einbezogen werden. Danach führt das Update Gate die beiden überarbeiteten Hidden States zusammen zum neuen Hidden State.

Das Problem des Vanishing Gradients bezieht sich auf das Gedächtnis des KI-Models. Das Problem tritt in den Trainingsphasen von neuronalen Netzen auf. Die Gradienten werden bei der Backpropagation immer kleiner, wodurch die betroffenen Schichten im Netzwerk fast gar nichts lernen. Die Gradienten werden kleiner, da eine Aktivierungsfunktion immer einen Wert zwischen 0 und 1 zurückgibt. Ein Gradient wird mit der Ableitung der Aktivierungsfunktion multipliziert, wodurch er gleichzeitig kleiner wird. Desto tiefer das Netzwerk ist, desto mehr Schichten müssen die Gradienten durchlaufen. Mit jeder Schicht wird der Gradient somit exponentiell kleiner. Die führt dazu, dass das betroffene KI-Modell sich keine Informationen Langzeitig merken kann. Bei GRUs wird das Problem

des Vanishing Gradients durch die Gates gelöst. Sie stellen eine gewichtete Mischung aus altem und neuen Hidden State dar. Dadurch wird der Gradient nicht ständig verkleinert und wichtige Informationen können über einen langen Zeitraum gespeichert werden.

4.2.5 Bidirektionale RNNs

Um ein noch besseres Ergebnis zu erhalten, kann man auch ein Bidirektionales RNN nutzen. Dieses besteht aus zwei RNNs. Eines liest die Zeitfenster von vorne und das andere von hinten ab. Dadurch bekommt man die doppelte Menge Hidden States und die gesamte Vorhersage wird sehr viel robuster.

$$\mathbf{h}_t = \left[\overrightarrow{\mathbf{h}}_t \,;\, \overleftarrow{\mathbf{h}}_t\right]$$

Auch in AMT-Systemen sind Bidirektionale RNNs sehr hilfreich, da musikalische Ereignisse auch eine klar verständliche Abhängigkeit, von der Zukunft in die Vergangenheit aus, haben. Das gleiche Prinzip kann man auch auf LSTMs oder GRUs einsetzen. Bidirektionale RNNs sind noch robuster, aber dafür ist der Rechenaufwand bei weitem höher. Ein Bidirektionales RNN kann man auch in folgender Arbeit finden. [16]

4.2.6 Die Geschichte von RNNs

Die Geschichte von Recurrent neural networks reicht bis in das Jahr 1990 zurück. [7] In dieser Arbeit wurde erstmals die Idee der Rückkopplung eingebaut, wodurch der Output eines Neurons als zusätzliche Eingabe im nächsten Zeitfenster genutzt wird. Das resultierte später in den Hidden States. Diese Arbeit baute den Grundstein für alle weiterführenden RNNs. Eine weitere wichtige Arbeit über RNNs ist die folgende. [20] Diese Arbeit über RNNs wurde, offiziell im Jahre 1997, publiziert, obwohl sie schon 1986 als ein technischer Bericht veröffentlicht war. Zudem wurde, im Jahre 1997, das erste Bidirektionale RNN [36] und das erste LSTM erfunden. [17] Erst über ein Jahrzehnt später wurde das erste GRU entwickelt. [5] RNNs fanden ihren Weg in die automatische Musiktranskription, fast zeitgleich zu CNNs, im Jahre 2015. [38] Für LSTMs dauerte die Einbindung in AMT-Systeme etwas länger. Erst im Jahre 2016 wurde ein LSTM in einem AMT-System eingefügt. [37] Dahingegen wurde das erste Bidirektionale RNN erst Ende 2017 in ein AMT-System integriert. [16] Noch ein Jahr später wurde das erste Mal auch ein GRU in einem AMT-System genutzt. [22]

4.3 Transformers

Transformer sind eine, von Google entwickelte Deep-Learning Architektur, die mit dem Prinzip der Self-Attention, zur natürliche Sprachverarbeitung (NLP) genutzt wird. Auch Transformer verarbeiten die Daten, wie ein LSTM, sequentiell. Jedoch wurde zu dieser Verarbeitung noch der Attentionmechanismus hinzugefügt, welcher der grundlegende Baustein eines Transformer-Modells ist. Die Architektur eines Transformers besteht grundlegend aus 3 Bausteinen. Diese 3 sind Input Embedding + Position Encoding, der Self-Attention Layer und der Feedforward Layer. Input Embedding und Position Encoding nur einmalig am anfang des Transformers durchgeführt. Dahingegen werden Self-Attention und Feedforward mehrmals hintereinander aufgerufen. Dies passiert meistens 6-, 12- oder 24-mal, wodurch das Transformer-Modell an Layer gewinnt. Dieser Schritt wird so oft wiederholt, damit der Transformer immer komplexere musikalische Strukturen erkennen kann. Als Nächstes erläutere ich die einzelnen Schritte eines Transformers näher.

4.3.1 Input Embedding und Position Encoding

Zunächst müssen die Daten, welche der Transformer verabeiten soll in das richtige Format für diesen umgewandelt werden. Dafür ist das Input Embedding zuständig. Tokens sind einzelne Datenpunkte wie zum Beispiel kleine Wörter wie "hallo". Wenn man zum Beispiel bei ChatGPT die Anfrage stellt "Wie alt sind die Planeten unseres Sonnensystems", dann wird zwar das Wort "alt" als ein Token gespeichert, aber längere Wörter wie "Sonnensystem" werden aufgeteilt in "Sonn" und "ensystem". Je nach Tokenizer-Version, die der jeweilige Transformer nutzt, kann dies jedoch abweichen. Unser

Beispielsatz nutzt, alleine für die Wörter, 9 Tokens. Tokens können auch einzelne Satzzeichen oder Symbole sein. Für verschiedene Transformer Modelle gibt es immer eine maximale Tokenanzahl, bei GPT-3.5 sind es um Beispiel ungefähr 4000 Tokens. Diese Tokens sind für den Input und Output. Heißt, wenn der Input zu viele Tokens nutzt, gibt es weniger Tokens für den Output. Dies ist jedoch meistens, wie man an der Menge der Tokens für unseren Beispielsatz sehen kann, kein größeres Problem. In AMT-Systemen stellen Tokens Zeitfenster (Frames) oder Musik-Events (Onsets etc.) dar. Jedoch können diese von einem Transformer nicht verarbeitet werden, weshalb sie durch Input Embedding erstmals in Vektoren umgewandelt werden. Dies passiert über eine Embedding-Matrix, welche die einzelnen Tokens in den Vektorraum einbettet.

$$Embedding - Matrix \in \mathbb{R}^{MaximaleTokenanzahl \times Embedding-Dimension}$$

Tokens, welche ähnliche musikalische Eigenschaften speichern, werden im Vektorraum näher beieinander gespeichert. Das Modell kann die Tokens richtig einordnen durch die gesammelten Daten im Training.

Nun sieht der Transformer die Tokens trotzdem nur als viele Vektoren, ohne Rheinfolge, an. Um den Tokens einen Sinn zu geben, müssen diese eine explizite Position erhalten. Das wird durch Position Encoding gemacht. Durch zwei Funktionen, eine Sinus und eine Cosinus, wird für jeden Token ein Positionsvektor berechnet. Dadurch kann der Transformer die relative Position, verschiedener Tokens, gut miteinander vergleichen und zugleich wiederkehrende Muster erkennen. Danach wird der Positionsvektor dem Tokenvektor aufaddiert. Moderne Transformer besitzen manchmal auch gelernte Position Embeddings, wodurch die Positionen schon durch das Training bekannt sind. Diese Vektoren werden, als Input, weiter an den Transformer geleitet.

4.3.2 Self-Attention Layer

Jeder Input-Vektor wird einzeln behandelt. Durch Self-Attention kann sich jetzt jeder Vektor merken, welche anderen Vektoren für sich selber relevant sind. Zunächst wird jeder Input-Vektor in 3 verschiedene Vektoren umgewandelt. Diese Vektoren heißen Query, Key und Value.

- **Query** (Q): Bestimmt, auf welche Informationen aus anderen Tokens das Modell aktuell achten möchte.
- **Key** (K): Zeigt anderen Tokens, was dieser Input-Vektor anderen Tokens, für Informationen, anbieten kann.
- **Value** (V): Enthält die tatsächlichen Informationen des Input-Vektors.

Als Nächstes wird der Attention Score berechnet. Durch diesen wird die Kompatibilität zu anderen Tokens berechnet. Als Ergebnis bekommt man, für jeden Token, eine Matrix. Jeder Wert in dieser Matrix steht für die Ähnlichkeit zwischen zwei verschiedenen Tokens. Diese Werte sind die Attention Weights. Durch Softmax werden diese Werte normalisiert. Als Letztes werden diese Attention Weights mit den Value-Vektoren verbunden.

$$\text{Output} = \text{Attention Weights} \times \text{Value Vektoren}$$

Nun können die einzelnen Tokens sich besser auf die Tokens fokussieren, welche wirklich wichtig für deren Ergebnis sind.

4.3.3 Feedforward Layer

Der Feedforward Layer stammt von der Idee eines Feedforward neuronalen Netzwerks (FNN). Dieses neuronale Netz verläuft immer nur in eine Richtung und hat keine Rückkopplung. Das heißt jeder Token wird für sich alleine, nacheinander, umgeschrieben. Das funktioniert so, da die jeweilige Gewichtung schon im Self-Attention Layer stattgefunden hat. Im Feedforward Layer passiert dann folgendes. Zunächst werden die Vektoren, welche wir aus dem Self-Attention Layer gewonnen haben,

mit einer Gewichtsmatrix auf eine höhere Dimension transformiert. Durch die erhöhung der Dimension bekommt das Netzwerk mehr Freiraum Informationen voneinander zu trennen und komplexere Zusammenhänge zu modellieren. Durch eine Aktivierungsfunktion, zum Beispiel ReLU, erlernt das Modell jetzt nichtlineare Abhängigkeiten. Diese Aktivierungsfunktion gehört zu dem Hidden Layer des FNN. Ein FNN kann mehrere Hidden Layer besitzen, welche alle die Input-Werte in irgendeiner Form verändern. Als Nächstes wird der Vektor wieder auf seine ursprüngliche Dimension zurückprojiziert, wodurch wir nur die wichtigsten Informationen behalten. Mathematisch sieht der Feedforward Layer folgendermaßen aus:

$$y_t = \text{FFN}(x_t) = (x_t W_1 + b_1) \xrightarrow{\text{ReLU}} W_2 + b_2$$

- x_t: Eingabevektor des Tokens.
- W_1, W_2: Gewichtsmatrizen, zuständig für Dimensionserweiterung und -reduktion.
- b_1, b_2: Bias-Vektoren der jeweiligen linearen Projektionen.
- $(x_t W_1 + b_1)$: Ergebnis der ersten linearen Projektion.
- ReLU: Aktivierungsfunktion zur Einführung von Nichtlinearität.
- y_t: Ausgabewert des Feedforward Layers für das Token.

4.3.4 Geschichte

Das erste Transformer-Modell wurde im Jahre 2017 erfunden. [41] Durch den Self-Attention Layer wurde die sequenzielle Modellierung von Daten revolutioniert. Leider dauerte es noch lange bis der erste Transformer auch in AMT-Systemen genutzt wurde. Das liegt an mehreren Gründen. Einerseits gibt es im gegensatz zu Natural Language Processing (NLP) viel weniger Datensätze zum Lernen. Transformer trainieren auf einem globalen Level und brauchen daher sehr viele Datensätze. Die Rechenkosten von Transformer sind auch weitaus höher als andere Systeme wie CNN und RNN. Zudem lag der Fokus bei AMT-Systemen für eine sehr lange Zeit überwiegend bei CNN und RNN basierenden Systemen, da diese Anwendungsfälle bekannter waren in AMT-Systemen. Das größte Problem war jedoch wahrscheinlich die Anpassung. Transformer Modelle waren einfach nicht dafür ausgelegt Musikspezifische Daten zu verarbeiten. In der Musiktranskription wird immer mit langen Sequenzen, ein gesamtes Musikstück, gearbeitet. Spezielle Transformer Modelle für diesen Anwednugnsfall mussten noch programmiert werden. Die ersten Ansätze für Transformer in AMT-Systemen wurden zwischen den Jahren 2021 und 2023 erstellt. Eine der ausschlaggebendsten Modelle war der Music Transcription Transformer (MT3), welches von Magenta/Google Brain erstmals im Jahre 2022 veröffentlicht wurde. [10] Dieses spielt eine große Rolle in der Transformer-basierten automatischen Musiktranskription, da es das erste weit verbreitete Multi-Task-Transkriptionsmodell ist. Eine der Letzten bedeutenden Errungenschaften zu Transformern und Musiktranskription ist das YourMT3+ Modell, indem die MT3-Architektur noch weiter verbessert wurde. [**chang2024yourmt3**]

4.4 Weitere Deep-Learning-Module und Entwicklungen

CNNs und RNNs oder Transformer Modelle wie MT3 werden überwiegend in AMT-Systemen eingebunden, wenn es sich um KI integration handelt. Jedoch gibt es noch andere KI-Modelle, die auch ab und zu in AMT-Systemen integriert werden. Diese gehen, bei der Musiktranskription, ganz anders vor als die vorgestellten Module. Je nach KI übernehmen sie auch eine ganz andere Aufgabe. Im folgenden Teil werde ich ein paar von diesen anderen KIs vorstellen. Dabei werde ich bei der am meisten erprobten KI anfangen, sodass die letzte KI welche ich vorstelle nur theoretisch, für die Musiktranskitpiton, besprochen word.

4.4.1 Variational Autoencoder

Variational Autoencoder (VAE) ist ein KI-Modell, welches komplexe Daten in eine verdichtete Form überführt und durch Wahrscheinlichkeiten bestimmte Muster vorhersagen kann. [23] Dabei ist wich-

tig das VAE kein alleinstehendes KI-Modell ist, sondern eher als zusatz gilt um bestimmte Bereiche zu verbessern. In der Musiktranskription könnte ein VAE zum Beispiel bei der Mehrdeutigkeit von Musik helfen. Wenn mehrere Instrumente spielen ist es schwer die genauen Noten herauszuschreiben. Da VAEs, anders als CNNs oder RNNs, als ausgabe keine eindeutigen Noten herausgeben, sondern eine Wahrscheinlichkeit, könnte dies helfen die entscheidung, welche Note gerade spielt, robuster zu gestalten. Dadurch das sich VAEs nur eine wahrscheinliche Version der Musik, aus den Daten, extrahieren könnte man auch kreative Variationen des Musikstückes damit bilden. Die Methode, welche in VAEs genutzt wird, fand Ihren Ursprung in folgendem Paper. [24]

4.4.2 Graph Neural Network

Graph Neural Network (GNN) ist ein KI-Modell, welches dazu dient Daten, mit abhängigkeit voneinander, zu verarbeiten. Diese Daten werden in einem Graphen aus Knoten und Kanten gespeichert. Dabei hat jeder Knoten seine eigenen Daten, die er in jedem Rechenschritt mit seinen benachbarten Knoten austauscht. Knoten sind benachbart, wenn sie durch Kanten verbunden sind. Dadurch wird jeder Knoten im Netzwerk schlauer. Natürlich machen RNNs und Transformer etwas Ähnliches wie ein GNN, mit zum Beispiel Backpropagation. Der Unterschied ist hier, das GNNs keine bestimmte Reihenfolge, wie Wissen weitergeleitet wird, haben. Es ist ein anderer Ansatz harmonische Abhängigkeiten miteinander zu kombinieren. [43]

4.4.3 Diffusion Model

Diffusion Models arbeiten mit Rauschen. Aus komplettem rauschen bauen sie Schritt für Schritt ein sauberes Musikstück zusammen. Dabei entscheiden sie pro Rechenschritt nur einen kleinen Teil des Musikstückes, wodurch mehrdeutige Passagen in Musikstücken auch noch später im Transkriptionsprozess behandelt werden können. In dem Projekt DiffRoll, aus dem Jahre 2022, wurde ein Diffusion Model auf ein AMT-System angewendet. [3]

4.4.4 Reinforcement Learning

Reinforcement Learning (RL) benutzen ein Belohnungssystem, damit der Agent sich beim training richtig anpasst. Sobald der Agent ein bestimmtes Ziel erreicht oder eine Handlung ausführt, wird dieser dafür belohnt. Die Agenten, welche am meisten Belohnungen erzielen werden kopiert und in der nächsten iteration eingesetzt, bis man einen Agenten besitzt, welche die gewünschte Aufgabe erfüllt. Dieses Prinzip wird vor allem in Videospielen, wo es meistens eindeutige Ziele gibt, eingesetzt. In der Musiktranskription könnte man dieses Prinzip folgendermaßen umsetzen. Man gibt dem Agenten als Input eine Audiodatei. Danach lässt man ihm Noten nacheinander erzeugen. Durch eine andere KI oder vorgefertigte MIDI-Dateien besitzt man einen Vergleichsdatensatz. Falls die gewählten Noten vom Agenten gleich dem Vergleichsdatensatz sind, bekommt der Agent Belohnungen. So kann er sich selber Musikalisches wissen aneignen und Transkriptionen von anderen KIs nochmal korrekturlesen. [29]

4.4.5 Energy-Based Model

Der letzte Ansatz ist ein Energy-Based Model (EBM). Ein EBM arbeitet mit Energie anstatt von Wahrscheinlichkeiten. [27] Die verschiedenen Arten, wie das EBM das Musikstück transkribieren kann haben alle jeweils ihre eigene Energie. Dabei haben die besten möglichkeiten immer die niedrigste Energie, wie bei dem Gradientenabstiegsverfahren. Dies ist für AMT-Systeme interessant, da das EBM somit viele verschiedene Transkriptionsraten wählen könnte. In mehr improvisierten Musikrichtungen, wie Jazz, bildet das eine größere viel fallt. Man kann auch Musiktheorie in das Modell mit einbauen, wodurch aus einem Musikstück viele Remixe kreiert werden können. Der Ansatz von EBMs in AMT-Systemen ist, von den zusätzlich vorgestellten KI-Modellen, wahrscheinlich einer der interessantesten. Leider sind EBMs sowohl schwer zu trainieren, als auch sehr rechenintensiv. AMT-Systeme ohnehin schon recht anspruchsvoll, weshalb noch kein EBM-Modell in der automatischen Musiktranskription zum einsatz kam.

5 KI-basierende AMT-Systeme im Vergleich

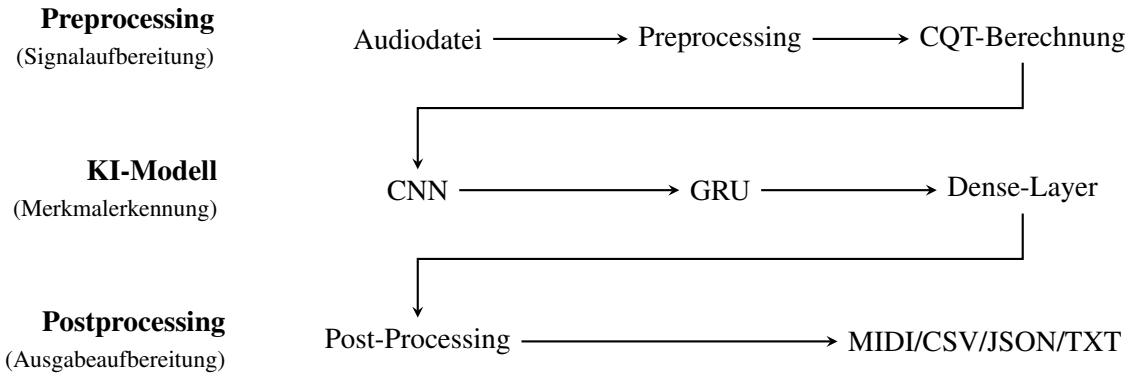
Im Laufe der Forschung zu AMT-Systemen wurden schon einige verschiedenen Architekturen eingesetzt und verbessert. Jedes neue System hat, unabhängig des KI-Moduls, eine komplett eigene Struktur und vorgehensweise. Im Verlauf dieser wissenschaftlichen Arbeit habe ich die Geschichte und viele Konzepte der automatischen Musiktranskription dargestellt. Deshalb wird sich jetzt dieses Letzte große Kapitel um die state of the art AMT-Systeme handeln. Dafür stelle ich zwei verschiedene AMT-Systeme vor. Diese sind das CNN + GRU basierte Omnizart und das Transformer basierte MT3. Diese nutzen unterschiedliche KI-Modelle. Ich werde deren Architektur, sowie deren stärken und schwächen, ausgiebig erläutern.

5.1 Omnizart

Zum einen haben wir Omnizart, welches CNNs und RNNs als KI-Modelle nutzt. [42] Der Name Omnizart setzt sich aus den Wörtern Omni (alles) und Mozart zusammen, da ihr Ziel darin liegt, so viele Arten von Musik wie möglich zu transkribieren. Je nach Anwendungsfall werden verschiedene KIs benutzt. Dabei ist der Aufbau dieser KIs meistens gleich. Alle KI-Modelle bestehen aus einem CNN und einem bidirektionalen GRU. Der Anwendungsfall, zum Beispiel Drums oder Melody, hat dabei nur Einfluss auf die Trainingsdaten und dem Output. Omnizart ist ein Open-Source-Toolkit für AMT. Dadurch kann man, je nach Bedarf, ein KI-Modell auswählen was perfekt auf eine bestimmte aufgabe ausgelegt wurde. Omnizart findet ihren Ursprung im Jahre 2020 am Music and Culture Technology Lab, National Taiwan University. Omnizart hat seit seiner Gründung keine ausschlaggebenden weiteren Technologien, in der richtung KI, hinzugefügt. Dafür gibt es viele verschiedene CNNs und RNNs und einen open source code, welchen man gut zur eigenen Forschung an AMT-Systemen nutzen kann.

Omnizart ist ein modulares AMT-System, welches auf einer Deep-Learning-Architektur basiert. Der Trainingssatz besteht aus CQT-Spektren. Die KI-Modelle lernen dabei durch Supervised Learning.

Die Piepline von Omnizart kann man in drei Schritte aufteilen:



Der erste Schritt ist die Vorverarbeitung und Merkmalsextraktion (Preprocessing). Dadurch wird die Inputaudiodatei in ein CQT-Spektrogramm umgewandelt. Zunächst wird das gegebene Audiosignal durch folgende Methoden standardisiert:

1. **Mono-Konvertierung:** Die KI benötigt keine räumlichen Informationen, weshalb der linke und rechte Kanal von Stereosignalen in ein Monosignal addiert werden.
2. **Normalisierung:** Der Wechsel von zu großen und kleinen Amplituden kann die KI überfordern und ungenaue Ergebnisse liefern, weshalb das Audiosignal durch normalisierung auf einen einheitlichen Lautstärkebereich gebracht wird.
3. **Resampling:** Unterschiedliche Abtastraten führen zu Verzerrung und Frequenzverschiebung, deshalb wird diese auf eine einheitliche Rate, passend zu dem genutzten Modul, gebracht.

4. **Trimming:** Falls am Anfang oder Ende des Audiosignals Stille ist, wird diese durch Trimming entfernt, sodass das KI-Modell nicht unnötig verwirrt wird.

Danach wird das standardisierte Audiosignal umgeformt zu einem CQT-Spektrogramm.

Im zweiten Schritt werden die KI-Modelle genutzt, um die Merkmale des Audiosignals vorherzusagen und zu extrahieren. Viele der AMT-Systeme, welche ich im Laufe dieser Arbeit vermerkt habe, besitzen eine ähnliche Architektur wie Omnizart. [16] Der Unterschied zu diesen AMT-Systemen ist das Omnizart, je nach Anwendungsfall, verschiedene Module nutzt. Omnizarts Module sind:

- **Chord:** Akkorderkennung
- **Drum:** Drum-Transkription
- **Melody:** Melodietranskription
- **Vocal:** Gesangsmelodietranskription
- **Piano:** Polyphone Klaviertranskription
- **Multi-Pitch:** Mehrstimmige Tonhöhenschätzung
- **Beat/Downbeat/Chord-Labelling:** Rhythmus, Takt & Akkorde

Jedes Modul bekommt als Input ein CQT-Spektrogramm. Dieses wird durch ein CNN verarbeitet, welches die Eigenschaften und Merkmale der Musik extrahiert. Mit diesen Daten modelliert dann ein GRU die zeitliche Abhängigkeit. Durch den Dense Layer werden die Ergebnisse des GRUs in zum Beispiel Onsets, Beats und viele weitere musikalischen Attribute, umgewandelt.

Im dritten Schritt wird der Output der KI-Modelle nochmals aufbereitet. Fehler werden zunächst durch folgende Methoden verbessert:

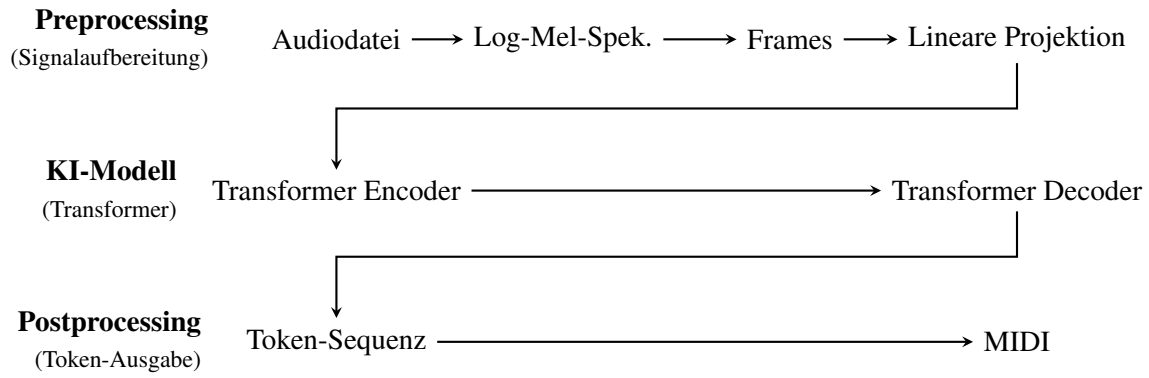
1. **Noten-Segmentierung:** Wenn derselbe Ton in zwei nacheinander folgenden Frame ein Onset hat, wir der zweite Onset gelöscht und der Note hinzugefügt, sodass keine Notendopplungen entstehen.
2. **Onset-Korrektur:** Falls ein Onset zeitlich nicht zur richtigen Zeit erfasst wurde, wird der Einschwingzeitpunkt des Onset an seinen Frame genau angepasst.
3. **Thresholding:** Noten, die nicht einen bestimmten Wahrscheinlichkeits-Threshold überschreiten, werden aussortiert.
4. **Quantisierung:** Je nach Taktstruktur können Noten zeitliche angepasst werden, sodass diese besser in zum Beispiel einen 3/4-Takt passen und das Stück rhythmischer ist.

Je nach Modul, welches man gerade nutzt, werden nur paar oder alle dieser Methoden eingesetzt. Auch deren Parameter unterscheiden sich je nach Modul. So hat das Drums-Modul ein viel kleineren Schwellwert als das Melodien-Modul bei Thresholding. Danach werden die Vorhersagen in vollständige Noten (Onset/Sustain/etc.) zusammengefasst und in das gewünschte Format übertragen. MIDI ist dabei das wichtigste Format, da dieses relevant für Musiksoftware ist, es gibt aber auch noch drei andere Formate die, je nach Modul, ausgegeben werden. Fast immer wird auch eine CSV-Datei ausgegeben. Darin befinden sich die Transkriptionsdaten, welche zur Analyse oder Forschung genutzt werden können. JSON-Dateien werden in den Chord- und Beat-Modulen ausgegeben. Dies liegt daran, dass JSON-Dateien verschachteltet Daten, wie Akkordfolgen über mehrere Takte, besser speichern können. In ihnen werden vor allem Takt-Informationen und Daten für Akkorde gespeichert. TXT-Dateien werden dahingegen nur wahlweise in Modulen genutzt. Sie sind ausschließlich für Debugging dar, weshalb sie für die meisten Nutzer nicht relevant sind. Die Daten werden, in einer TXT-Datei, in einer unstrukturierten Liste ausgegeben.

5.2 MT3

Das Transformer-Modell, welches ich näher erläutere, heißt MT3. MT3 steht für "Multi-Task Multitrack Music Transcription". MT3 ist gleichzeitig der Name für das Transformer-basierte KI-Modell, als auch für das überliegende AMT-System. Diese beiden sind untrennbar voneinander, da sie zu einer End-to-End-Architektur gehören. Das MT3-Modell bildet einen wichtigen Meilenstein für die Transformer-basierte automatische Musiktranskription. MT3 ist das erste weit verbreitete Multi-Task-Transkriptionsmodell, das heißt verschiedene Transkriptionsaufgaben werden von einem einzigen Modell gelöst. Frühere Modelle brauchten zum Beispiel für Drums und die Melody, wie es bei Omnizart der Fall ist, verschiedene KI-Modelle. Durch die Communityversion YourMT3+ wird dieses KI-Modell immer weiter gepflegt und verbessert.

MT3 ist ein Transformer-Modell, welches spezifisch für Musiktranskription entwickelt wurde. Es besteht grundlegend aus folgenden Bereichen:



Als Input wird eine Audiodatei bereitgestellt. Meistens geht man mit dem Datentyp WAV (Waveform Audio File Format), da dieser verlustfrei und standardisiert ist. Das Audiosignal wird dann mit STFT analysiert und als Spektrogramm ausgegeben. Mithilfe von Librosa wird aus diesem ein Log-Mel-Spektrogramm erzeugt. Librosa ist ein Python Paket, welches wichtige Methoden für Musik und Audio analyse bereitstellt. Dieses Log-Mel-Spektrogramm wird nun in Frames aufgeteilt. In dem MT3-Modell sind diese Frames meist 32ms lang. jeder Frame besitzt N Mel-Frequenzbänder, welche die Dimensionsgröße dieses Frames bestimmen. Um jetzt für jeden Frame eine einheitliche Dimension zu bekommen, mit der die KI Arbeiten kann, wird lineare Projektion genutzt. Lineare Projektion ist ein Matrix-Multiplikator, welche ein Frame auf eine, für das KI-Modell, normalisierte Dimension bringt, zum Beispiel 512 oder 1024. Daraus entstehen Input Embeddings, mit denen die KI nun arbeiten kann.

Jetzt folgt der Encoder, welcher das normale Prinzip eines Transformers, mit Self-Attention Layer und Feedforward Layer, ausführt. In dem MT3-Modell werden, vor der ausführung der Layer, die Positionen der Frames durch Positional Embeddings festgelegt. In MT3 besteht der Encoder aus 6 vollständigen Transformer-Layern. Der Inhalt jedes Frames wird jetzt zusammenhängen mit allen anderen Frames gesetzt. Daraus resultieren Vektoren, welche kontextreiche Informationen des Musikstückes besitzen. Diese Vektoren werden weiter an den Decoder geleitet

Aus den gegebenen Vektoren extrahiert der Decoder die wichtigen Daten. Neben den Encoder Vektoren als Input bekommt dieser zudem autoregressive Tokens. Autoregressive Tokens sind bereits generierte Tokens, die im Training meist aus den echten Daten stammen (Teacher Forcing) und beim Einsatz vom Modell selbst generiert werden. Dadurch bekommt der Decoder mehr Kontext für die zu generierenden Tokens. Neben den normalen Tokens bekommen auch die autoregressiven Tokens Positional Embeddings aufaddiert. Bevor der Decoder nun durchläuft, kann man mithilfe von Task-Conditioning, einen Task-Token hinzufügen. Das MT3-Modell transkribiert immer polyphon, jedoch kann man durch den Task-Token trotzdem den Output auf einen bestimmten Task, wie zum Beispiel Klavierstücke oder Trommelnoten, auslegen. Durch vorheriges Lernen ist das MT3-Modell darauf

ausgelegt, nach einem Task-Token die folgenden Tokens in einer bestimmten Art und Weise zu transkribieren. Der Ablauf des Decoders sieht folgendermaßen aus:

1. **Self-Attention:** verarbeitet den Kontext der autoregressiven Tokens
2. **Cross-Attention:** extrahiert relevante Informationen aus dem Encoder-Output
3. **Feedforward Layer:** verfeinert die Repräsentationen der Tokens lokal
4. **Lineare Layer:** wandelt den Vektor in ein konkretes Token um

Dabei besteht der Decoder, im MT3-Modell, auch aus 6 vollständigen Transformer-Layern. Jeder Token stellt einen Teil eines musikalischen Events dar. Diese heißen zum Beispiel "Note-On C4" oder "Shift +10 ms". Ein musikalisches Event wäre somit eine volle Note, mit Notennamen, Onset und Offset, Sustain und weiteren musikalischen Eigenschaften, die man auf einer Note anwenden kann. Als Output gibt der Decoder jeden einzelnen Token zurück.

Jetzt werden im Post-Processing die Tokens zu Sequenzen (musikalischen Events) zusammengesetzt. Durch einige weitere Tools werden die Noten zudem noch bereinigt und verbessert.

- **Zeitberechnung:** Die Shift-Tokens werden aufsummiert, sodass alle Noten zur richtigen Zeit abgespielt werden.
- **Noten validierung:** Jede Note wird darauf geprüft, ob sie ein On- und Offset besitzt.
- **Velocity Korrektur:** Manche Noten haben keine oder eine fehlerhafte Anschlagstärke, welche bei diesem Schritt im Nachhinein hinzugefügt oder überschrieben wird.
- **Fehlerbereinigung:** Unlogische Notenfolgen, wie zwei Shifts hintereinander, werden entfernt.

Am Ende werden die bereinigten Noten in einem standardisierten Musikformat, als MIDI-Datei, ausgegeben.

5.3 Omnizart vs MT3

Die beiden vorgestellten AMT-Systeme sind grundlegend verschieden aufgebaut. Jetzt ist die Frage, welche dieser beiden Systeme ist besser geeignet und unter welchen Voraussetzungen sollte man sich für welches System entscheiden.

Der größte Unterschied in der Architektur liegt darin, das Omnizart verschiedenen KI-Modelle für unterschiedliche Aufgaben besitzt, während MT3 ein einziges leistungsstarkes KI-Modell besitzt. Somit kann man bei Omnizart leichter einen bestimmten Task verbessern oder analysieren. Das ist vor allem gut, wenn man sich auf monophone Musikstücke konzentriert und einen einfacheren Einstieg in die automatische Musiktranskription bekommen möchte. Zudem gibt Omnizart eine größere Menge an Outputdaten zurück als MT3, bei dem nur eine MIDI-Datei ausgegeben wird. MT3 hingegen besitzt ein einziges KI-Modell, wodurch man nicht gezielt einen speziellen Task umstrukturieren kann. Dafür eignet sich das KI-Modell direkt wissen der verschiedenen Tasks an un kombiniert dieses. Somit können polyphone Musikstücke deutliche besser transkribiert werden. Durch YourMT3+ gibt es zudem weiteres Ausbaupotential, dahingegen entwickelt sich Omnizart nicht sonderlich weiter.

Bei den KI-Modellen hat MT3 einen klaren Vorsprung. CNNs und GRUs sind schon seit längerem in AMT-Systemen vertreten. Sie wurden ausgiebig angepasst und verbessert für diesen Anwendungsfall. Hingegen zu diesen KI-Modellen ist das MT3 erst seit paar Jahren im Rennen. Durch YourMT3+ wird es immer weiter entwickelt, wodurch es in der nahen Zukunft zu einem Standard der automatischen Musiktranskription werden könnte. Zudem ist das MT3-Modell deutlich besser auf polyphone Musikstücke ausgelegt. In der mehrheit von Audiosignalen gibt es mehrere Stimmen. Die meisten Menschen möchten auch lieber eine einfache Lösung, die wenig eigenaufwand benötigt. Deshalb wäre ein zentrales KI-Modell, zur transkribierung von allen verschiedenen Audiosignalen, die populärste Lösung. Ein Schwachpunkt des MT3-Modells ist der Rechenaufwand. Da alles über ein KI-Modell

läuft, muss dieses umso mehr Rechenschritte durchführen und braucht exponentiell mehr GPU auslastung und Speicherkapazität.

Omnizart ist empfehlenswert, falls man sich gerade mit dem Forschungsgebiet neu auseinandersätz. Man sieht viel schneller Fortschritte und kann sich erstmals auf kleiner KI-Modelle konzentrieren. In den meisten anderen Fällen würde ich jedoch das MT3-Modell, oder das Nachfolgermodell YourMT3+, empfehlen. Dieses ist der State of the Art und wird auch noch in einigen Jahren Support erhalten. Zudem ist man bei diesem Modell in keiner Weise eingeschränkt und kann jegliche Musik transkribieren. Dieses Modell geht jedoch auch mit viel Rechenaufwand und einem guten Verständnis des Forschungsgebiets einher. Deshalb sollte man sich, bevor man das MT3-Modell nutzt, gut damit auseinandergesetzt haben.

6 Fazit

Abschließende Bemerkungen, Reflexion und Ausblick.

Literaturverzeichnis

[1] Leonard E Baum u. a. “A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains”. In: *The annals of mathematical statistics* 41.1 (1970), S. 164–171.

[2] Sebastian Böck und Markus Schedl. “Polyphonic Piano Note Transcription with Recurrent Neural Networks”. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Zugriff am 19.05.2025. 2012, S. 121–124. URL: https://www.cp.jku.at/people/schedl/Research/Publications/pdf/boeck_schedl_icassp_2012.pdf.

[3] Kin Wai Cheuk u. a. “DiffRoll: Diffusion-based Generative Music Transcription with Unsupervised Pretraining Capability”. In: *ICASSP*. arXiv preprint arXiv:2210.05148, Oct 2022. Apr. 2023.

[4] Kin Wai Cheuk u. a. “Revisiting the onsets and frames model with additive attention”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, S. 1–8.

[5] Junyoung Chung u. a. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014).

[6] smith dave und wood chet. “the ’usi’, or universal synthesizer interface”. In: *journal of the audio engineering society* 1845 (Okt. 1981).

[7] Jeffrey L Elman. “Finding structure in time”. In: *Cognitive science* 14.2 (1990), S. 179–211.

[8] Florian Eyben u. a. “Universal onset detection with bidirectional long-short term memory neural networks”. In: (2010).

[9] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological cybernetics* 36.4 (1980), S. 193–202.

[10] Josh Gardner u. a. “MT3: Multi-task multitrack music transcription”. In: *arXiv preprint arXiv:2111.03017* (2021).

[11] AP Goswami und Makarand Velankar. “Study paper for Timbre identification in sound”. In: *International Journal of Engineering Research & Technology (IJERT)* 2.10 (2013).

[12] Alex Graves, Santiago Fernández und Jürgen Schmidhuber. “Multi-dimensional recurrent neural networks”. In: *International conference on artificial neural networks*. Springer. 2007, S. 549–558.

[13] Xiangming Gu u. a. “Automatic lyric transcription and automatic music transcription from multimodal singing”. In: *ACM Transactions on Multimedia Computing, Communications and Applications* 20.7 (2024), S. 1–29.

[14] Xiangming Gu u. a. “Deep audio-visual singing voice transcription based on self-supervised learning models”. In: *arXiv preprint arXiv:2304.12082* (2023).

[15] Yoonchang Han, Jaehun Kim und Kyogu Lee. “Deep convolutional neural networks for predominant instrument recognition in polyphonic music”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.1 (2016), S. 208–221.

[16] Curtis Hawthorne u. a. “Onsets and frames: Dual-objective piano transcription”. In: *arXiv preprint arXiv:1710.11153* (2017).

[17] Sepp Hochreiter und Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), S. 1735–1780.

[18] iZotope. *What is the Noise Floor?* Zugriff am 19.05.2025. n.d. URL: https://www.izotope.com/en/learn/what-is-the-noise-floor.html.

[19] Fatemeh Jamshidi u. a. “Machine learning techniques in automatic music transcription: A systematic survey”. In: *arXiv preprint arXiv:2406.15249* (2024).

[20] Michael I Jordan. “Serial order: A parallel distributed processing approach”. In: *Advances in psychology*. Bd. 121. Elsevier, 1997, S. 471–495.

[21] S Johanan Joysingh, P Vijayalakshmi und T Nagarajan. “Development of large annotated music datasets using HMM based forced Viterbi alignment”. In: *TENCON 2019-2019 IEEE Region 10 Conference (TENCON)*. IEEE. 2019, S. 1298–1302.

[22] Minju Jung, Haanvid Lee und Jun Tani. “Adaptive detrending to accelerate convolutional gated recurrent unit training for contextual video recognition”. In: *Neural Networks* 105 (2018), S. 356–370.

[23] Diederik P Kingma, Max Welling u. a. “An introduction to variational autoencoders”. In: *Foundations and Trends® in Machine Learning* 12.4 (2019), S. 307–392.

[24] Diederik P Kingma, Max Welling u. a. *Auto-encoding variational bayes*. 2013.

[25] Anssi Klapuri und Antti Eronen. “Automatic transcription of music”. In: *Proceedings of the Stockholm Music Acoustics Conference*. 1998, S. 6–9.

[26] Yuta Kusaka und Akira Maezawa. “Mobile-AMT: Real-Time Polyphonic Piano Transcription for In-the-Wild Recordings”. In: *2024 32nd European Signal Processing Conference (EUSIPCO)*. IEEE. 2024, S. 36–40.

[27] Yann LeCun u. a. “A tutorial on energy-based learning”. In: *Predicting structured data* 1.0 (2006).

[28] Yann LeCun u. a. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), S. 541–551.

[29] Juncheng Li u. a. “Music theory inspired policy gradient method for piano music transcription”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[30] Magenta Team. *Performance RNN: Generating Expressive Piano Performances with Neural Networks*. `https://magenta.tensorflow.org/performance-rnn`. Accessed: 2025-06-22. 2017.

[31] Lukáš Samuel Marták, Rainer Kelz und Gerhard Widmer. “Balancing bias and performance in polyphonic piano transcription systems”. In: *Frontiers in Signal Processing* 2 (2022), S. 975932.

[32] Keith D. Martin. *Automatic Transcription of Simple Polyphonic Music: Robust Front End Processing*. Techn. Ber. Technical Report 399. Zugriff am 19.05.2025. MIT Media Lab, 1996. URL: `https://www.media.mit.edu/publications/automatic-transcription-of-simple-polyphonic-music-robust-front-end-processing-2/`.

[33] James A. Moorer. “On the Transcription of Musical Sound by Computer”. In: *Computer Music Journal* 1.4 (1977). Zugriff am 19.05.2025, S. 32–38. URL: `https://www.jamminpower.org/PDF/JAM/Transcription%20of%20Musical%20Sound%20-%20CMJ%201977.pdf`.

[34] Konstantinos Pyrovolakis, Paraskevi K. Tzouveli und Giorgos Stamou. “Multi-Modal Song Mood Detection with Deep Learning”. In: *Sensors* 22.2 (2022), 358? DOI: `10.3390/s22020358`.

[35] Eric D Scheirer. “Tempo and beat analysis of acoustic musical signals”. In: *The Journal of the Acoustical Society of America* 103.1 (1998), S. 588–601.

[36] Mike Schuster und Kuldip K Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11 (1997), S. 2673–2681.

[37] Siddharth Sigtia, Emmanouil Benetos und Simon Dixon. “An end-to-end neural network for polyphonic piano music transcription”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24.5 (2016), S. 927–939.

[38] Siddharth Sigtia u. a. “A hybrid recurrent neural network for music transcription”. In: *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2015, S. 2061–2065.

[39] Haruto Takeda u. a. “Hidden Markov model for automatic transcription of MIDI signals”. In: *2002 IEEE Workshop on Multimedia Signal Processing.* IEEE. 2002, S. 428–431.

[40] Yohannis Telila, Tommaso Cucinotta und Davide Bacciu. “Automatic music transcription using convolutional neural networks and constant-q transform”. In: *arXiv preprint arXiv:2505.04451* (2025).

[41] Ashish Vaswani u. a. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[42] Yu-Te Wu u. a. “Omnizart: A general toolbox for automatic music transcription”. In: *arXiv preprint arXiv:2106.00497* (2021).

[43] Zonghan Wu u. a. “A comprehensive survey on graph neural networks”. In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), S. 4–24.