technische universität
dortmund

Masterarbeit

Modellierung des menschlichen Musikgeschmacks mit neuronalen Netzen

Fabian Ostermann
9. Juni 2021

Gutachter:
Prof. Dr. Günter Rudolph
Dr. Igor Vatolkin

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Algorithm Engineering (LS11)
http://ls11-www.cs.tu-dortmund.de

Inhaltsverzeichnis

1. Einleitung

„Es gibt zwei Arten von Musik. Gute Musik und die andere Art."
Duke Ellington (1899–1974)

Wir verwenden unsere Urteilskraft des guten Geschmacks alltäglich. Stets wählen wir aus einem überwältigenden Angebotsreichtum intuitiv unsere Favoriten, sei es im frühen Kindesalter die Lieblingsfarbe oder im Erwachsenenalter das liebste Restaurant. Beim „durchzappen" durch die Vielzahl an Radiosendern haben wir nach wenigen Sekunden einen Impuls des Ge- oder Missfallens. Geschmack hilft uns die Objekte in unserer Welt zu ordnen, wenngleich diese Ordnung eine individuelle und höchst subjektive Angelegenheit ist.

Geschmack ist aber nicht nur Hilfsmittel des Konsumenten. Er ist in selbem Maße wichtig für den Schöpfer, der ein Objekt des Gefallens kreiert. Musikgeschmack ist die Richtwaage eines jeden Musikschaffenden, um das eigene Werk nach Schönheit und Sinnhaftigkeit beurteilen zu können. Ein Komponist am Klavier mit Notenblatt und Stift, oder heute auch am Computer mit einem Musikprogramm, besitzt doch lediglich das Werkzeug seines individuellen Geschmacks, um seine eigene Leistung zu bewerten. Doch was ist, wenn der Komponist kein Mensch, sondern eine Maschine ist?

Algorithmen und Maschinen in das Handwerk der Komposition zu integrieren ist kein neuartiger Ansatz. Bereits vor dem Computerzeitalter wurden Algorithmen entworfen, die kompositionelle Arbeiten systematisieren sollten. Seit Computer zur universellen Komplexmaschine geworden sind, gibt es Versuche, vollautomatische Kompositionssysteme zu implementieren. Ihre Erzeugnisse stehen dabei seit jeher unter großer Kritik und werden gemeinhin als „schlecht" konnotiert. Woran liegt das?

Von den großen Komponisten wird erwartet, dass ein veröffentlichtes Werk einen Mindeststandard an Qualität erfüllt. Dieser Mindeststandard wird dabei am verallgemeinerten Gefallen gemessen. Was oftmals übersehen wird: Auch den großen Köpfen misslingt etwas. Ihr eigener Geschmack rät dann zur Überarbeitung oder gar zum Neuanfang. Das endgültige Werk ist also meist das Ergebnis mehrerer Versuche. Benötigt der Komponist wenige Versuche, oder gelingt ihm jeder Versuch mit Leichtigkeit zu einem Meisterwerk, spricht man gemeinhin vom musikalischen Genie [vgl. Jacob, 1996, S.1].

Der algorithmische Komponist hat (bisher) keinerlei automatisiertes Maß, um seine Erzeugnisse zu bewerten. Optimiert der Entwickler sein System auf möglichst „gute" Erzeugnisse, so ist der „gute Geschmack" impliziert. Die Algorithmen werden unter der Prämisse entworfen, dass jedes generierte Erzeugnis „gut" sein soll. Wäre dies erfolgreich, wäre der Computer der vorangegangenen Argumentation zufolge ein Genie. Ferner bedeutet dies, all solche Algorithmen werden im Ansatz so entworfen, dass ein Genie entstehen soll. Das ist offenkundig eine überambitionierte Aufgabenstellung. Und tatsächlich: Computer als Künstler sind (zur Zeit) eher unteres Mittel-

maß. Kein aktueller Ansatz kann mit einem professionellen menschlichen Komponisten mithalten [vgl. Fernandez u. Vico, 2013, S.561].

Geht man davon aus, dass mit steigender Zahl der Versuche prinzipiell die Zahl der guten Ergebnisse steigt, haben Computer dem Menschen etwas voraus: Sie sind „enorm fleißig“. Wie Collins [2018] in bestechender Weise mit seinem *Billion Song Dataset* zeigt, können Computer mit Leichtigkeit eine enorme Menge an Musikstücken hervorbringen. Problematisch ist einzig die Qualität dieser Stücke. Reiner „Ideenreichtum“ genügt offenbar nicht. Doch könnten sie gute von eher schlechten Resultaten trennen, würden sie unmittelbar zu einem besseren Künstler.

Ein strukturelles Problem ist die Art und Weise, wie Arbeiten in dem Bereich der automatisierten Musikgenerierung angelegt und durchgeführt werden. Es sind häufig Einzelstücke von einzelnen Entwicklern, gerne auch als Nebenprojekt, die wenig dokumentiert sind und Funktionen ohne Schnittstellen mit geringer Wiederverwendbarkeit besitzen [Fernandez u. Vico, 2013, S.559]. Allein das „zum laufen bringen“ der Software stellt deshalb oftmals eine große bis unüberwindbare Hürde dar.

Diese Arbeit möchte das Konzept des abgeschlossenen Musiksystems überwinden. Deshalb werden fortan Musikprogramme, seien es komponierende Algorithmen, intelligente Klangerzeuger oder automatische Bewertungsroutinen, als *künstliche* Komponisten, *künstliche* Musiker und *künstliche* Zuhörer bezeichnet, um das multiplikative Potential ihrer Interaktion zu verdeutlichen.[1]

Die rhetorische Personifizierung der Programme soll der Veranschaulichung der Grundidee dieser Arbeit als die *Metapher vom künstlichen Musikproduzenten* dienen. Musikproduzenten haben in der Musikindustrie die Aufgabe, Musikstücke zu hören und ihr Potential (meist im Hinblick auf kommerziellen Erfolg) einzuschätzen. Beispielsweise entscheiden sie, welche Bewerber einen Plattenvertrag bekommen, helfen Bands Stücke für ihr nächstes Album auszuwählen oder machen Verbesserungsvorschläge während der Produktion eines Songs. Sie benötigen einen erfahrenen Musikgeschmack und Nutzen ihn zur Klassifikation von Musik in einem professionellen Umfeld.

Die Metapher des *künstlichen Produzenten*, die in ihrer konkreten Umsetzung zu Anfang des zweiten Teils dieser Arbeit ausgeführt wird, führt im nächsten Abschnitt zur pragmatischen Annahme, dass der Musikgeschmack als binäres Klassifikationsproblem aufgefasst werden kann. Anschließend wird das Forschungsgebiet des *Machine Listening* vorgestellt, dass sich mit der Thematik des Verstehens von Musik durch Algorithmen beschäftigt. Die darauffolgenden beiden Abschnitte rechtfertigen die Wahl von neuronalen Netzen als Klassifikationsmethode und diskutieren, worauf im Kontext von Musikdaten und mit Hinblick auf den Stand aktueller Forschung besonderer Augenmerk gelegt werden soll. Am Schluss der Einleitung wird die genaue Zielsetzung und Gliederung der Arbeit vorgestellt.

1.1. Musikgeschmack als binäres Klassifikationsproblem

Die vorliegende Arbeit möchte überprüfen, ob und wieweit sich der menschlich-subjektive Musikgeschmack mithilfe neuronaler Netze, also mit Methoden des maschinellen Lernens, vorhersagen lässt. Damit sollen automatisierte Vorhersagen über die allgemeine oder subjektive Bewertung

[1] Dies ist eine Anlehnung an den Begriff *künstliches Leben*. Rowe [2001] als auch Collins [2018] weisen darauf hin, dass allein die Veränderung des sprachlichen Gebrauchs limitierende Denkgewohnheiten aufbrechen kann.

sowie grundsätzliche Aussagen über die Qualität von Musikstücken möglich sein. Verwendet werden Klassifikatoren, um den Output bestehender automatischer Musikgeneratoren (*künstliche Komponisten*) zu selektieren und die „Perlen ihrer Arbeit“ herauszusuchen (*künstlicher Musikproduzent*). Dies stellt als eine nachträgliche Optimierung bereits bestehender Ansätze eine *Metaoptimierung* dar.

Ein Klassifikator ist eine Funktion $f(\mathbf{x}) = y$ mit $y \in \{1, \ldots, k\}$. Der Eingabevektor $\mathbf{x}$ ist der Merkmalsvektor, also eine beliebig geformte Repräsentation des Eingabeobjektes. Methoden des maschinellen Lernens haben die Aufgabe, die Funktion f zu generieren. Dazu werden ihnen Eingabeobjekte gegeben, deren Klassen $\hat{y}$ bekannt sind. Man spricht von Beispielen $(\mathbf{x}, \hat{y})$ einer Trainingsmenge. [Goodfellow u. a., 2016, S.99,100,105]

Der Musikgeschmack g soll in dieser Arbeit als solch ein Klassifikationsproblem eines beliebigen Musikstücks M repräsentiert durch das Eingabeobjekt $\mathbf{m}$ in die Klassen $\smile$ („gefällt“) und $\frown$ („gefällt nicht“) aufgefasst werden.[2] Ein *künstlicher Musikproduzent* trifft mithilfe einer gelernten Geschmacksfunktion g eine binäre Entscheidung:

$$g(\mathbf{m}) = y \qquad \text{mit } y \in \{\smile, \frown\} \tag{1.1}$$

Als Eingabevektor $\mathbf{m}$ können verschiedene digitalisierte Repräsentationen von M genutzt werden. Sie leiten sich im Kontext dieser Arbeit aber stets direkt aus der Hörerfahrung des Musikstücks ab. Darüber hinaus werden keine weiterführenden Metainformationen zugeführt.

1.2. Machine Listening

Die Forschungsfelder im Spannungsfeld zwischen Computern und Musik sind vielfältig. Neben der Informatik und den Musikwissenschaften spielen meist auch Psychologie, Physik, Ingenieurswissenschaften, Statistik oder Wissensmodellierung eine Rolle. Ein ordnender Überblick über einige der sich überschneidenden Forschungsgebiete soll in diesem Abschnitt gegeben werden.

Das Forschungsgebiet der *Computer Audition* [Wang, 2011] (Analogon zu *Computer Vision*) widmet sich der grundlegenden Aufgabe Computern das „Hören“ zu ermöglichen. Es will die auditive Wahrnehmung des Menschen modellieren und simulieren, um dem Computer ein Verständnis für Schallereignisse zu implementieren. Früher stand vor allem das Übermitteln, Aufnehmen und Manipulieren von Audiosignalen im Mittelpunkt der Technologieentwicklung. Heute steigt der Anteil und die Relevanz analysierender und reaktiver Systeme. Sprachassistenzsysteme sind vom anfänglichen Expertentool [vgl. Klatt, 1977] zum praktikablen Alltagsgerät [vgl. Stift. Warentest, 2019] gereift: Unsere Geräte hören uns zu!

Und es geht weiter: Unsere Geräte hören auch Musik! Sie ermitteln für uns Songtitel und Interpreten in Sekundenschnelle [Wang, 2003] oder schlagen uns Songs vor, die uns gefallen [Jannach u. a., 2010]. Maschinelle Klassifikationsverfahren übernehmen immer komplexere Aufgaben auf höherem Abstraktionsniveau: Sie werden „intelligenter“. Klassifikation von Musik ist typischerweise im Forschungsgebiet des *Music Information Retrieval* (MIR) [Collins, 2010, S.248ff] verortet. Es behandelt Themen wie Genre-Klassifikation, Instrumentenkennung, automatische Transkription oder Empfehlungssysteme, aber auch die automatische Musikgenerierung.

[2] Das Symbol $\smile$ ist eine Stilisierung von ☺; $\frown$ dementsprechend von ☹.

MIR kann mit Ausnahme der Musikgenerierung einerseits als Musik-bezogenes Teilgebiet der *Computer Audition* gesehen werden und ist andererseits gerade durch die Verbindung zur Generierung grundlegend für diese Arbeit. Dabei ordnet sie sich dem Teilgebiet des sogenannten *Machine Listening* (manchmal auch *Automatic Listening*) [Collins, 2010, S.113ff] zu. Dieses beinhaltet die intelligente Verknüpfung von gewonnen Informationen sowie den flexiblen Umgang damit. Frühe Untersuchungen auf diesem Gebiet setzten vor allem Heuristiken ein. Nach der Popularität von vermeintlich objektiveren statistischen Ansätzen stehen heute vornehmlich Methoden der künstlichen Intelligenz (KI) und des maschinellen Lernens im Mittelpunkt.

Die Idee dem Computer ein Maß für den musikalischen Höreindruck zu implementieren ist nicht neu. Berger [2001] stellte bereits ein Modell zur automatischen Perzeption von Musik durch Algorithmen vor und war in diesem Bereich damit Vordenker. Er versuchte das Moment von Erwartung und Überraschung in einer Komposition zu formalisieren. Maurer [1999] erkannte das Potential für selbst-evaluierende Musikgeneratoren. Der komponierende Algorithmus hätte dann Zugriff auf eine „Kritikfunktion“ (ähnlich der Idee von Gleichung (1.1)). Dieser Ansatz deckt sich mit der in der Einleitung eingeführten Forderung: *Künstliche Musiker* und *künstliche Komponisten* benötigen einen Sinn für Geschmack (Klassifikator), um in einer menschlichen Könnensprüfung bestehen zu können.

Collins [2018] beschreibt in einem futurologischen Essay wie eine qualitative Erfahrbarkeit und Evaluierung durch Computermusiksysteme eine stetige Qualitätsverbesserung ihrer Resultate verursachen könnte. Er formalisiert seine Ideen in einem Konzeptentwurf einer futuristischen (halb-)automatischen Musikproduktion [Collins, 2018, S.11]. Der (unendliche) Optimierungszyklus enthält drei Akteure: Zunächst erstellt ein *künstlicher Komponist* ein Musikstück (oder mehrere) auf Grundlage einer ausgewählten Musiksammlung. Dazu geben menschliche Zuhörer ihre Meinungen ab. Ein *künstlicher Zuhörer* lernt mittels Methoden des maschinellen Lernens diese Meinungen zu antizipieren. Dann trifft er aus allen existierenden Musikstücken eine Auswahl, die eine neue Musiksammlung bilden. Der künstliche Komponist erstellt auf deren Grundlage wiederum neue Musikstücke.

Zugleich thematisiert Collins die Grenzen einer vermeintlich objektiven Qualitätsbestimmung. Er weist darauf hin, das Computermusik ebenso dem subjektiven Gefallen unterworfen ist wie menschengemachte Musik. Er fordert die Forschenden auf, die Motivationen hinter ihren Bestrebungen zu hinterfragen. Die Diversität an Musikgeschmäckern und die Vielzahl an unterschiedlichen Definitionsversuchen sei nicht Problem, das eines Konsens bedarf, sondern ein logisch begründbarer Umstand, der akzeptiert werden müsse, um in der Sache weiter zu kommen [vgl. Collins, 2018, S.15]. Musik ist ein essentiell menschliches Phänomen [Collins, 2018, S.3]. Es besteht großes Potential im Automatisieren von menschlichen Kreativprozessen [vgl. Collins, 2018, S.4].

Allerdings sind Computer vom Wesen her zunächst sehr verschieden zu den Menschen. Es bestehe nach Collins [vgl. 2018, S.8] ebenso eine Gefahr, wenn das Verhalten von Algorithmen voreilig vermenschlicht wird. In der Motivation ein Programm auf allgemeingültige Erfahrungswerte aufzubauen, würden unter Umständen bereits im Design unüberwindbare Schranken unbewusst fest eingebaut.

Der Autor der vorliegenden Arbeit folgert aus den dargelegten Überlegungen, dass einem Algorithmus eine gewisse Selbstständigkeit und unabhängige Entscheidungsgewalt überlassen

werden sollte, wenn es darum geht, neue Wege zu entdecken. Dies ist ein wichtiger Grund für die Wahl des adaptiven maschinellen Lernverfahren der künstlichen neuronalen Netze. Weitere Gründe werden im folgenden Abschnitt diskutiert.

1.3. Warum künstliche neuronale Netze?

Das menschliche Gehirn wurde von den Neurowissenschaften als Netzwerk einer enormen Zahl untereinander verbundener Neuronen identifiziert (vgl. Abbildung 1.1). Neuronen sind Nervenzellen, die über chemische Synapsen untereinander Signale verschicken [Kandel u. Schwartz, 1995]. Das einzelne Neuron ist demnach ein Schalter. In starker Vereinfachung und Abstraktion dienen sie in der Informatik als Modell eines trainierbaren maschinellen Klassifikators. Technische Grundlagen folgen in Kapitel 2. Hier sollen zunächst die Möglichkeiten und Schranken ihrer Anwendung im vorliegenden Kontext diskutiert werden.

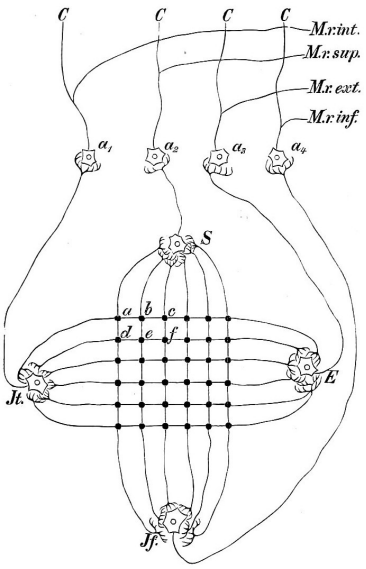


Abbildung 1.1.: Erste schematische Darstellung eines neuronalen Netzes.
[Grafik aus Exner, 1894, S.193]

Künstliche neuronale Netze (KNN) simulieren die Funktionen des menschlichen Gehirns. Vor allem für Anwendungen, die annähernd auf dem Niveau der menschlichen Kognitions- und Antizipationsleistung arbeiten sollen, werden sie erfolgreich eingesetzt. Man könnte meinen: Ist das Netz groß, die Rechengeschwindigkeit schnell und die Trainingsbeispiele zahlreich genug, so müsste ein simuliertes Gehirn mit komplex-simulierten Denkprozessen ähnliche Intelligenz besitzen wie ein Mensch. Davon ist aktuelle Forschung aber noch weit entfernt.

Bereits Berger [vgl. 2001, S.280] befand, KNNs könnten die Mechaniken menschlicher Kognitionsprozesse, die menschliche Vorhersagen steuern, auf flexible Art und Weise simulieren und visualisieren und seien so im Kontext von künstlicher Musikkomposition der notwendige Schlüssel, um die kreativen mentalen Prozesse nachzubilden. Durch ihre Komplexität besitzen KNNs eine Vielzahl an Parametern. Das bedeutet, dass es aktuell noch viele Ansätze gibt, die Netze innerhalb verschiedenster Anwendungsgebiete zu optimieren. Das Forschungsfeld *Computer Audition* hat hier noch viele offene Forschungsfragen. Die MIR-Forschung bleibt eher noch ein Stück weiter zurück. Im Vergleich zur Domäne der Bildverarbeitung bleibt die Audio- und vor allem die Musikdomäne in ihrer Erforschung von KNNs zurück. Das ist tatsächlich ein typisches Verhältnis, das über die Geschichte des Forschungsgebiets hinaus für viele weitere Gebiete gilt [vgl. Hutmacher, 2019]. Ein Grund dafür ist, dass Musik weniger intuitiv greifbar ist als Bilder: Der Mensch ist *Augenmensch*. Darüber hinaus ist notwendiges Fachwissen zu Audiotechnik und Musiktheorie weit weniger verbreitet. Zuletzt besitzen visuelle Medien auch ein höheres kommerzielles Potential.

Überwachtes Trainieren eines KNNs ist im Kontext dieser Arbeit insofern von Vorteil, dass die entscheidenden Merkmale nicht im Vorhinein bekannt sein müssen. Die Gründe für ein individuelles Geschmacksurteil sind oftmals so unbewusst wie vielfältig. Das Netz lernt selbstständig Prinzipien, die dem Urteil potentiell zugrunde liegen. Wird ein identisches Netz mehrmals auf

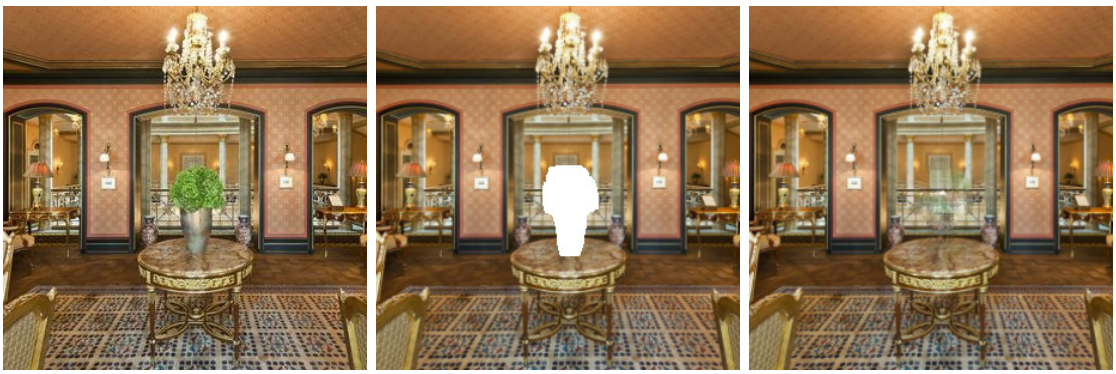
verschiedene Geschmäcker trainiert, das heißt die Urteile über die Musikstücke sind unterschiedlich auf die Klassen ⌣ und ⌢ verteilt, kann das Netz während des Trainings unterschiedliche „Prinzipien“ für seine Entscheidung ausprägen. Im Kontext von Klassifikationsproblemen konnte gezeigt werden, dass KNNs dazu im Stande sind, komplexe Muster zu erlernen und zu erkennen, auch ohne dass vorher eine klassische Regelabstraktion im Sinne einer Vorverarbeitung, Merkmalsgewinnung und Merkmalsreduktion erfolgen muss [vgl. Niemann, 1983]. Ein neuronales Netz erkennt Merkmale selbstständig und kann sogar zur Merkmalsgewinnung genutzt werden [Li u. a., 2010].

1.4. Grenzen künstlicher neuronaler Netze

Geht man davon aus, dass Algorithmen eine gewisse Schöpfungskraft oder eigene Kreativität haben können, bleibt die große Frage, inwieweit diese Kreativität wirklich etwas originell neues schaffen kann [Loughran u. O’Neill, 2017]. Typischerweise basieren Konzepte des Maschinenlernens in ihrem Kern (nur) auf der Analyse statistischer Verteilungen (theoretischer Hintergrund folgt in Kapitel 2). Das bedeutet für die Generierung von Ausgaben, dass sie lediglich eine Variation der gelernten Trainingsdaten sind. Sie „imitieren“ ein Beispiel wie es auch in der Trainingsmenge vertreten gewesen sein könnte. Dieser Umstand beschränkt die generelle Aussagekraft von neuronalen Netzen zur Generierung und zur Klassifikation in selbem Maße. Zur anschaulichen Erläuterung soll im Folgenden die Interpretation von Ergebnissen einer Arbeit [Iizuka u. a., 2017] zum Problem der Bildvervollständigung (engl. *Image Completion*) dienen.

Das mittlere Bild in Abbildung 1.2 soll vervollständigt werden. Ein KNN wurde dazu auf einer großen Menge ähnlicher Bilder trainiert. Das Ergebnis im rechten Bild zeigt, es konnte eine sinnvolle Ergänzung generiert werden und das in durchaus beeindruckender Weise. Das linke Originalbild zeigt jedoch, dass ursprünglich eine Blumenvase auf dem Tisch zu sehen war. Der Algorithmus hat keinen Hinweis darauf und versagt konsequenterweise. Er beweist durch die Generierung eines leeren Tisches aber noch mehr: Er hat kein kreatives Verständnis für seine Aufgabe, sondern versucht sich lediglich im Imitieren der vorgegebenen Umgebung.

In Abbildung 1.3 scheint das zunächst anders. Der Algorithmus vermutet zurecht, dass die beiden weißen Flecken im oberen Stockwerk Fenster sein könnten und zeichnet sie sinnvoll ein.



(a) Original (b) Eingabe (c) Ausgabe

Abbildung 1.2.: Automatische Bildvervollständigung auf Basis von KNNs.
[Grafik aus Iizuka u. a., 2017, Abb.9]

(a) Eingabe (b) Ausgabe

Abbildung 1.3.: Unerkannte Strukturen bei der automatischen Bildvervollständigung.
[Grafik aus Iizuka u. a., 2017, Abb.10]

Der Algorithmus hat aber kein Verständnis dafür, dass ein Gebäude typischerweise Fenster hat und diese möglicherweise gleichmäßig und symmetrisch angeordnet sind. Der Algorithmus erkennt lediglich die bekannten Träger über den fehlenden Fenstern. Das beweist der Ausschnitt unten rechts: Der Algorithmus antizipiert hier nicht das fehlende Fenster, sondern zeichnet eine Mauer. Er erkennt nicht die übergeordnete Struktur der Fensteranordnung mit drei Fenstern links und drei Fenstern rechts der Tür.

Übergeordnete Strukturen sind allerdings in der Musik von entscheidender Bedeutung. Das macht Musik für KNNs zu einem enorm schweren Anwendungsfall. Das grundlegende Problem algorithmisch komponierter Musik ist (immer noch), dass es ihr an globaler Koherenz und Struktur fehlt [Pons, 2018; Roberts u. a., 2018].

Abbildung 1.4 zeigt ein weiteres Problem, dass sich zwischen den beiden vorherigen verorten lässt. Zu sehen ist ein Hund. Das ist auch im mittleren Bild gut zu erkennen. Die Vervollständigung misslingt jedoch vollkommen. Das Bild verliert jegliche Integrität. Ein leerer Tisch ist als Resultat akzeptabel und realistisch. Ein halber Hund jedoch ist im Kontext realistischer Bilder eindeutig als falsch zu bezeichnen.

Muster und damit statistische Verteilungen werden selbst bei vermeintlich einfachen Problem schnell überkomplex. Die KI wirkt oftmals erstaunlich clever und im nächsten Moment plötzlich erstaunlich dumm. Zugegeben: In Abbildung 1.2 eine Blumenvase zu erwarten ist schlicht unfair. Auch Menschen könnten die Vase nicht vorhersagen (vor allem nicht ihr korrektes Erscheinungs-

(a) Original (b) Eingabe (c) Ausgabe

Abbildung 1.4.: Gravierender Fehler bei der automatischen Bildvervollständigung.
[Grafik aus Iizuka u. a., 2017, Abb.16]

bild). Aber Abbildung 1.4 zeigt, dass auch für den Menschen leicht lösbare Aufgaben die von KI händelbare Komplexität leicht übersteigen. Diese anschaulichen Beispiele zur Bildvervollständigung zeigen grundsätzliche Probleme von KNNs und lassen sich analog in die Musikdomäne übertragen. Auch im Kontext von Musikdaten lösen die aktuellen Erfolge von KNNs mitunter positives Erstaunen aus. Die Problemstellungen sind aber meist bei genauerer Betrachtung durch clevere Herangehensweisen vereinfacht worden. Das muss bei den aktuellen Einschränkungen der Intelligenz auch Teil der Zielsetzung sein. Collins [2018, S.10] meint sogar generell, dass wir von einer rationalen KI gar nicht erwarten könnten, dass sie Musik in einer allumfassenden Weise verstünde.

Daraus folgt, dass es umso wichtiger ist, genau abzuschätzen, was man von solch einer begrenzten KI wie einem KNN erwarten kann, um die vorgesehene Lösung maßgerecht auf die konkrete Problemstellung zuzuschneiden. Deshalb beschränkt diese Arbeit die Trainingsprobleme auf die Verbesserung künstlich generierter Musik von jeweils einem einzigen künstlichen Komponisten. Die Alternative wäre, ein einziges Netz zu trainieren, dass die Resultate aller möglichen Komponisten bewerten kann. Unter Eindruck der obigen Beispiele wäre dies jedoch eher kein erfolgversprechender Ansatz. Die Stücke eines einzelnen Komponisten weisen eine begrenzte statistische Varianz auf. Die Flexibilität und Adaptivität der KNNs könnte so ausreichend sein, um Vorhersagen von Geschmacksurteilen selbstständig aus Trainingsbeispielen abzuleiten.

1.5. Unerklärliche Effizienz

Für den Menschen ist ein Geschmacksurteil zunächst frei von Anstrengung und geschieht nahezu automatisch. Es ist leicht zu sagen, was uns gefällt. Umso schwerer fällt uns überraschenderweise oft die Antwort darauf, warum uns etwas gefällt [Gebesmair, 2001, S.11].

Mit KNNs, die das menschliche Gehirn modellieren wollen, ist es kurioserweise ziemlich ähnlich. Es ist derzeit meist schwer erklärbar, geschweige denn automatisch bestimmbar, warum KNNs fast allen anderen Ansätzen den Rang ablaufen [vgl. Goodfellow u. a., 2016, S.8]. Die Untersuchung fertiger Netze ist stets aufwändig und von Fall zu Fall höchst unterschiedlich. Lange wurden tiefgreifende Evaluationen sogar mehrheitlich vermieden [Lapuschkin u. a., 2016].

Mittlerweile gibt es einige Ansätze diese wichtige Lücke zu schließen (Details dazu in Abschnitt 2.4). Bisher sind diese aber wieder vornehmlich auf die Bildverarbeitung als primäres Anwendungsgebiet optimiert. Diese Arbeit will auch hier Möglichkeiten in der Anwendung auf Musikdaten vorstellen und untersuchen. Nur so kann erkannt werden, auf welchem Abstraktionsniveau die Netze tatsächlich arbeiten. Gerade im Kontext von Musikgeschmack, dessen grundlegende Prinzipien selbst nicht umfassend geklärt sind, sind Rückschlüsse auf die zugrundeliegende Arbeitsweise der trainierten Netze von besonderem Interesse.

1.6. Zielsetzung und Gliederung der Arbeit

Grundlegendes Ziel der Arbeit ist es, mithilfe von automatischen Klassifikatoren die Qualität der Resultate zweier Musikgeneratoren zu optimieren. Dabei soll die Frage beantwortet werden, ob und inwieweit KNNs ein gewinnbringendes Verfahren für dieses Vorhaben darstellen. Die

vorliegende Arbeit ist dazu in drei Teile gegliedert. Der erste Teil befasst sich mit den theoretischen Grundlagen, die im zweiten Teil in die Praxis umgesetzt werden. Im Schlussteil werden die Erkenntnisse zusammengefasst.

Das folgende Kapitel 2 erläutert Grundlagen zu digitalisierten Musikdaten als auch zum maschinellen Lernen mit KNNs. Dabei werden beide Themen gebündelt mit wechselseitigem Bezug zueinander betrachtet. Dazu dient auch ein historischer Überblick über diese Verbindung. Danach folgt in Kapitel 3 die Einführung von Methoden der algorithmischen Komposition, die in dieser Arbeit zwar nicht implementiert, aber zur späteren Analyse der Klassifikationsergebnisse benötigt werden, um korrekte Interpretationen zu ermöglichen. Dabei wird vor allem der Unterschied zwischen heuristisch regelbasierten Systemen und datenbasierten KI-Methoden besonders beleuchtet.

Der praktische Teil startet in Kapitel 4 mit der konkreten Einführung der *Metapher des künstlichen Musikproduzenten*. Eine dreistufige Entwicklung des bisherigen Verhältnisses zwischen Komponisten und Musikproduzenten wird beschrieben. Eine vierte Stufe wird im Rahmen dieser Arbeit als logischer nächster Schritt vorgeschlagen. Zudem wird beschrieben, inwieweit sich dieser Ansatz zu verwandten Ansätzen abgrenzt. Kapitel 5 stellt dann die Funktionsweise der zwei für diese Arbeiten gewählten Musikgeneratoren (*künstliche Komponisten*) vor und begründet die Wahl im Vergleich mit verwandten Arbeiten. Es wird zudem beschrieben, wie aus den vorhandenen Vorarbeiten zwei Datensätze für das Trainieren der KNNs gewonnen werden. Kapitel 6 schildert dann das Vorgehen der eigentlichen Experimentreihe. Dazu werden zunächst die Parameter der Eingabedaten und Netzarchitekturen beschrieben. Anschließend werden die Trainingserfolge evaluiert. In der Auswertung werden die unterschiedlich aufgebauten Modelle miteinander verglichen und die besten werden weiter optimiert, um zum Schluss das erfolgreichste Netz für jeden Datensatz zu identifizieren. Kapitel 7 widmet sich dann der tiefen Analyse und Interpretation der Netze. Die Erkenntnisse werden auch auf musiktheoretischer Ebene betrachtet. Zudem werden neue Formen der Visualisierung vorgeschlagen. Der praktische Teil der Arbeit schließt in Kapitel 8 mit einer Anwendungsstudie. Ziel ist es dabei, mithilfe der bis dato erfolgreichsten Netze ein (möglichst) „perfektes“ Stück zu generieren.

Der dritte Teil zieht ein Fazit. So erfolgt eine rückblickende Zusammenfassung der wichtigsten Erkenntnisse und es wird ein Ausblick über potentielle Anschlussarbeiten gegeben.

Teil I.

Grundlagen

2. Künstliche Neuronale Netze und Musik

Obgleich die aktuell große Popularität von KNNs auf die Erfolge der nahen Vergangenheit zurückzuführen sind, ist der konzeptionelle Ansatz keineswegs eine neue Erfindung. Die ersten Arbeiten sind in die 1940er zurückzuführen, damals noch bekannt unter dem Schlagwort *Kybernetik*. Die heute als *Deep Learning* bekannten Ansätze erlebten ihre zweite große Welle als *Konnektionismus* in den 1980–1990er Jahren. [vgl. Goodfellow u. a., 2016, S.13]

Für einen allgemeinen Überblick über wegbereitende Arbeiten sei auf Goodfellow u. a. [2016, Abschnitt 1.2] verwiesen. Diese Arbeit konzentriert sich auf die konkrete Verbindung von KNNs und MIR. Da der praktische Teil auf den Vergleich verschiedener KNN-Strukturtypen abzielt, werden in diesem Kapitel die populärsten Ansätze in ihrer chronologischer Einordnung vorgestellt.

Ein allgemeines Schema der Musterklassifikation im Maschinenlernen teilt sich in die Phasen Eingabe der Daten, Vorverarbeitung, Merkmalsgewinnung, Merkmalsreduktion, die eigentliche Klassifikation und die Ausgabe der Klassenvorhersage [Richard O. Duda, 2001, S.10]. Anders als bei klassischen Ansätzen der Musterklassifikation entfällt bei KNNs (meist) die klassische Merkmalsgewinnung und -reduktion. Stattdessen werden die Daten in der Vorverarbeitung in unterschiedliche Repräsentationen überführt. Die Wahl der Repräsentation hat dabei einen sehr großen Einfluss auf den Lernerfolg eines KNN [vgl. Goodfellow u. a., 2016, S.3]. Mögliche Repräsentationen von Musikdaten werden im folgenden Abschnitt 2.1 erläutert. Welche unterschiedlichen Netztypen es gibt und auf welche Weisen sie Eingaben verarbeiten, behandelt der anschließende Abschnitt 2.2. Abschnitt 2.3 erklärt die mathematischen Prinzipien des Trainings der Netze. Das trainierte Netz bleibt dabei zunächst eine Blackbox. Abschnitt 2.4 behandelt deshalb Möglichkeiten der Analyse fertiger Netze.

2.1. Eingaberepräsentationen

Identifiziert man Schallereignisse als physikalische Wellenformen, die mithilfe von Mikrofonen eingefangen und von Lautsprechern wiedergegeben werden können, dann sind diese Wellen als sogenannte *Audiosignale* die direkteste aller möglichen Repräsentationen. Um Audiosignale im Computer digital speichern und verarbeiten zu können, werden sie in einem Analog-Digital-Wandler diskretisiert. Daraus lassen sich spektrale Repräsentationen ableiten, wie der folgende Abschnitt 2.1.1 zeigt. Im Kontext von Musik ist zudem die Notenschrift Ausgangspunkt für weitere Repräsentationen. Diese können als symbolische Repräsentationen bezeichnet werden und finden in Abschnitt 2.1.2 Erklärung. Abschnitt 2.1.3 stellt weitere Abstraktionen der Repräsentationen durch Metamerkmale vor. Möglichkeiten der Überführung ineinander werden in Abschnitt 2.1.4 diskutiert.

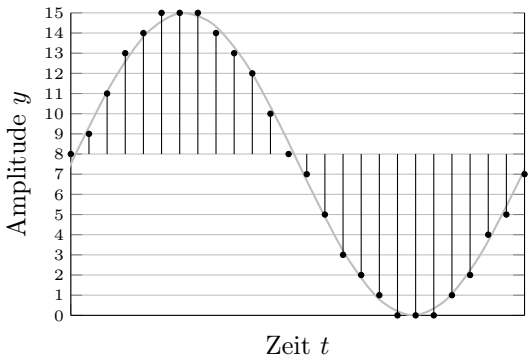


Abbildung 2.1.: Quantisierung einer Wellenfunktion. Mit konstanter Abtastrate wird die Amplitude y in $b = 16$ diskrete Werte überführt.

2.1.1. Audiosignale

Ein Audiosignal ist zunächst eine eindimensionale Schallwellen-Funktion $a(t) = y$ wobei t die Zeit und y die Amplitude bezeichnet. Zur Aufzeichnung wird der Schalldruck auf eine Membran durch ein Mikrofon (in der einfachsten Form durch elektromagnetische Induktion) in eine proportional pulsierende Gleichspannung übersetzt. Wird diese Spannung wiederum durch die Spule eines Lautsprechers geleitet, wird die Lautsprechermembran (aufgrund der Lorentzkraft) in einer nahezu identischen Schwingungskurve bewegt und die Schallwelle wird wieder hörbar.

Das Audiosignal $a(t)$ ist in der physikalischen Welt als Analogsignal zeit- und werte-kontinuierlich. Zur Überführung in einen digitalen Speicher muss das Signal quantisiert werden. Das ist Aufgabe eines *Analog-Digital-Wandlers*, der alle Δt Zeiteinheiten den anliegenden Spannungswert y aufzeichnet. Das Audiosignal wird in eine Folge von Amplitudenwerten $a = (a_0, \ldots, a_{N-1})$ mit einer endlichen Anzahl b an Möglichkeiten übersetzt (Abbildung 2.1). Eine Audiodatei ist also im einfachsten Fall eine Liste von Amplituden $a_k \in \mathcal{N}^b$ mit zigtausenden Einträgen pro aufgenommener Sekunde.

Abtastrate $r = \frac{1}{\Delta t}$ (auch Samplingrate, engl. *sampling rate*) und Samplingtiefe b (auch Bittiefe, engl. *bit depth*) bestimmen die Qualität der Digitalisierung. Aufgrund des Abtasttheorems ist die maximal abbildbare Tonfrequenz f_{max} eines digitalisierten Signals mit $f_{max} = \frac{r}{2}$ beschränkt [Shannon, 1949]. Der Dynamikumfang einer Aufnahme ist durch b beschränkt.

Fourier-Transformation

Die Fourier-Transformation (FT) beschreibt ein eindimensionales Signal als Komposition von Sinusschwingungen aller möglichen Frequenzen sowie deren zeitlicher Verschiebung. Ihre diskrete Form (DFT)

$$\hat{a}_j = \frac{1}{N} \sum_{k=0}^{N-1} a_k W_N^{-kj} \quad \text{mit } W_N = e^{2\pi \mathrm{i}/N} \tag{2.1}$$

stellt die Sinus- und Kosinusfunktionen kompakt in der komplexen Exponentialfunktion von W_N als Polarkoordinaten dar (eulersche Formel):

$$e^{\mathrm{i}\phi} = \cos(\phi) + \mathrm{i}\sin(\phi) \tag{2.2}$$

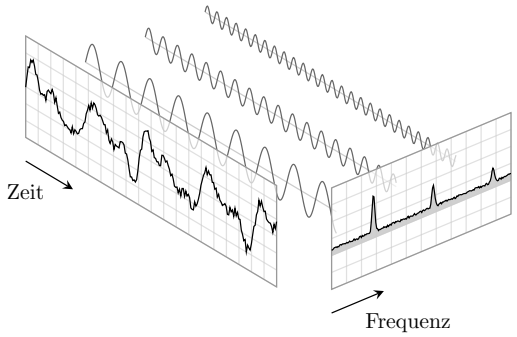


Abbildung 2.2.: Veranschaulichung der FT. Die Wellenfunktion (links) wird in ihre Sinusanteile zerlegt. Drei Sinusschwingungen verschiedener Frequenzen sind exemplarisch eingezeichnet. Die Funktion aller Sinuswerte ist das *Frequenzspektrum* (rechts).

Das ursprüngliche Signal a_k wird in die Fouriertransformierte $\hat{a}_j$ transformiert. Eine Veranschaulichung des Prinzips findet sich in Abbildung 2.2.

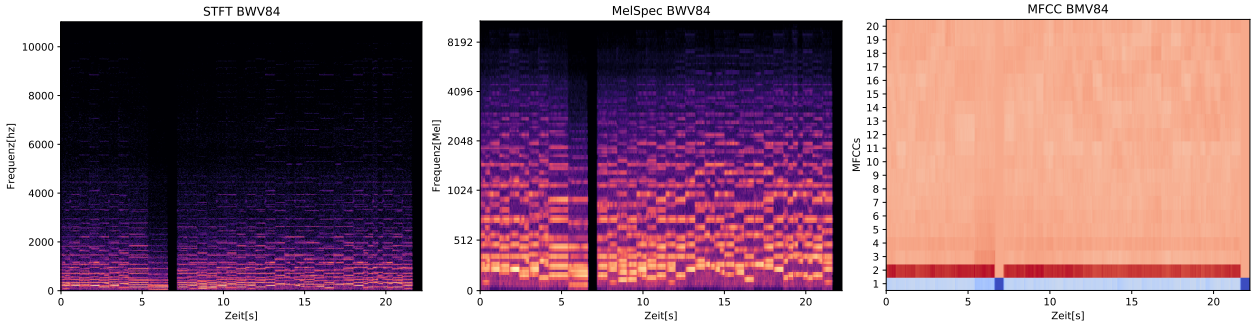
Da die Veränderung der Frequenz eines Sinustons vom Menschen als Tonhöhenveränderung wahrgenommen wird, spricht man von Frequenzanteilen eines Signals. Jedes Audiosignal liefert demnach einen komplexen Höreindruck aus unterschiedlichsten Sinustönen mit verschiedenen Tonhöhen, identifiziert als Grundtöne mit Obertönen beziehungsweise als Klangfarben (*Timbre*). Man nennt $\hat{a}_j$ auch *Spektralfunktion*. Toningenieure sprechen meist vom *Frequenzspektrum*.

Die FT hat in ihrer diskreten Form eine effiziente Berechnungsmöglichkeit: Die sogenannte schnelle Fourier-Transformation (engl. *fast Fourier transform*, FFT) hat einen Zeitaufwand von $\mathcal{O}(n \log n)$. Ausführliche und mathematische Ausführungen finden sich in Butz [2015].

Kurzzeit-Fourier-Transformation

Das Frequenzspektrum ermöglicht Aussagen über die Klangeigenschaften eines Audiosignals. Bei der Transformation in den Frequenzbereich verliert sich jedoch die zeitliche Information des Signals (Abbildung 2.2, rechts). Abhilfe schafft eine ausschnittsweise Betrachtung des Signals durch *Fensterung*. Der zeitliche Verlauf des Frequenzspektrums wird durch Aneinanderreihung von Signalausschnitten erkennbar. Solch eine Transformation wird als Kurzzeit-Fourier-Transformation (engl. *short-time Fourier transform*, STFT) bezeichnet. Sie stellt die einfachste Art eines *Spektrogramms* auf Basis einer DFT dar. Abbildung 2.3a zeigt als Beispiel die STFT einer Orgelaufnahme.

Die Fensterung bringt zwei grundlegende Probleme mit sich: Erstens staucht die Fensterung das ursprüngliche Signal um den Wert seines Versatzes. Durch Überlappung der Fenster kann dieser Effekt zwar vermindert werden, doch das führt wiederum zu Redundanzen in aufeinanderfolgenden Spektren. Für eine bessere Verlaufsbeschreibung sollte die Fensterbreite demnach möglichst klein gewählt werden. Die *küpfmüllersche Unschärferelation* besagt jedoch, dass das Produkt der Auflösungen im diskreten Zeit- und Frequenzbereich konstant ist [Küpfmüller u. Kohn, 1932, S.558–560]. Das bedeutet, die Verkleinerung der Fensterbreite führt zu einer geringeren Anzahl an einzelnen Frequenzwerten im Spektrum und umgekehrt. Die Wahl dieser Auflösungsrelation ist also ein wichtiger Parameter im jeweiligen Anwendungsfall.



(a) STFT (b) Mel-Spektogramm (c) MFCC

Abbildung 2.3.: Unterschiedliche spektrale Repräsentationen einer Orgelaufnahme von J. S. Bachs „Ich bin vergnügt mit meinem Glücke“ (BWV84). Man erkennt in allen Darstellungen die stille Pause bei 6,5 s.

Das zweite Problem ist eine Verzerrung des Frequenzspektrums durch den *Leck-Effekt* [Harris, 1978, S.52ff]. Die Fourier-Transformation beschreibt ein Signal durch Sinusfunktionen unendlicher Fortführung. Transformiert man ein endliches Signal wird es quasi als Endlosschleife betrachtet. Die Ränder der Fenster stellen sich dann als Sprünge dar, die im eigentlichen Signal nicht vorhanden sind, aber in der Transformation durch entsprechende Frequenzanteile dargestellt werden und das Spektrum somit verfälschen. Minderung dieses Effekts verschafft das „Ausblenden“ der Fensterränder durch Multiplikation des Signals mit einer *Fensterfunktion* [Butz, 2015, S.71ff], im einfachsten Fall zum Beispiel mit der cos^2-Funktion. Die Sprünge an den Rändern werden dabei fließend in Nullübergänge überführt.

Mel-Spektrogramm

In hörpsychologischen Untersuchungen konnte festgestellt werden, dass die Sensibilität für unterschiedliche Tonhöhen des menschlichen Hörempfindens nicht linear zur Frequenz ist. So sind tiefe Töne bereits bei geringerer Abweichung unterscheidbar als hohe Töne. Durch weiterführende Experimente wurde die Mel-Skala als Beschreibung abgeleitet [Stevens u. a., 1937]. Sie wird durch folgende Formel hinreichend approximiert [vgl. O'Shaughnessy, 1987, S.150]:

$$Z = 2595 \cdot \log_{10}(1 + \frac{f}{700}) \tag{2.3}$$

Die Frequenzanteile einer STFT sind zunächst äquidistant verteilt. Zur Überführung in die Mel-Skalierung werden die Frequenzwerte zu Mel-Werten (den sogenannten *Mels*) zugeordnet, wobei die Gewichtung über Dreiecksfilterung erfolgt. Abbildung 2.4 veranschaulicht dieses Prinzip. Aufgrund des Logarithmus bewirkt die Umskalierung eine Linearisierung im Sinne des menschlichen Hörempfindens. Abbildung 2.3b zeigt ein über die Zeit verlaufendes Mel-Spektrogramm. Da Musik auf dem menschlichen Hörempfinden fußt, wird diese Skalierung zur Analyse von Musik als sinnvoll befunden. Viele Arbeiten zeigen erfolgreiche Anwendungen. Zudem wird die Größe des Frequenzspektrums vor allem in den hohen Frequenzen ohne Informationsverlust reduziert. Dies ist von Vorteil bei der Klassifikation durch Maschinenlernen.

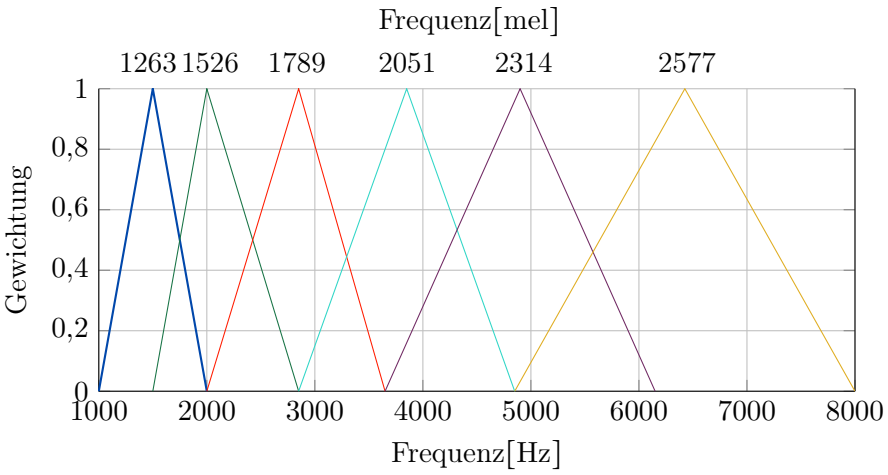


Abbildung 2.4.: Zuordnung der Frequenzen in Hz zu *Mels* per Dreiecksfilterung.

Mel-Frequenz-Cepstrum-Koeffizienten

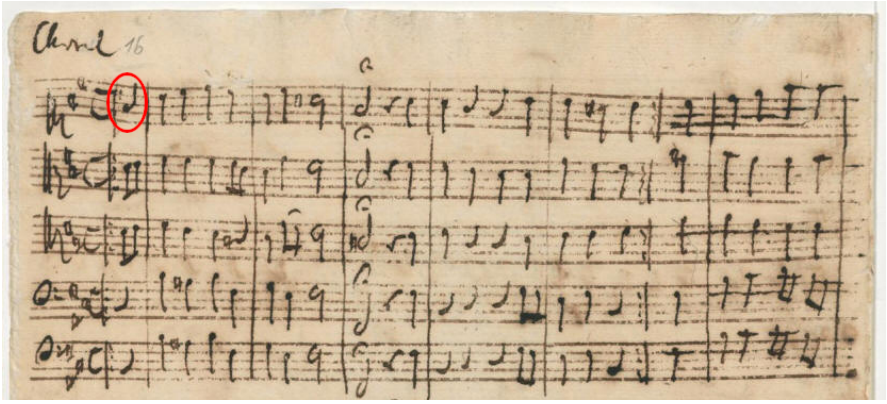
Mel-Frequenz-Cepstrum-Koeffizienten (MFCCs) sind eine Weiterentwicklung der Idee des *Cepstrums* [Bogert u. a., 1963]. Das Cepstrum ist das Spektrum vom logarithmierten Spektrum, also eine weitere DFT der logarithmierten Spektralfunktion $\hat{a}_j$ (notwendigerweise in ihrer inversen Form [Butz, 2015, S.97]). Die Logarithmierung dient hier ursprünglich ebenso der Linearisierung des Tonhöhenempfindens. Es liegt nahe diesen Schritt durch die Mel-Skalierung zu erweitern. Die inverse DFT wird zusätzlich noch durch die diskrete Kosinustransformation ersetzt, um die durch Dreiecksfilterung entstandenen Überlappungen zu dekorrelieren. [vgl. Logan, 2000]

So wie das Spektrum die Klangeingenschaften eines Signals analysiert, analysieren MFCCs die Eigenschaften des Spektrums. Besonders vorteilhaft ist dabei die Abstraktion vom Grundton auf die bloße Obertonstruktur des Klangs. MFCCs sind deshalb seit jeher besonders erfolgreich in Anwendungen der Spracherkennung [O'Shaughnessy, 1987], die unabhängig der Tonhöhe eines Sprechers Vokale und Konsonanten identifizieren sollen. Für die Anwendung auf Musik bedeutet dies, dass weniger die kompositionellen Eigenschaften der Stücke sondern eher deren Klangerscheinung beschrieben wird. Je nach Anwendungsfall kann das von Vor- oder Nachteil sein. Ein Vorteil für die Anwendung im Maschinenlernen ist die starke Reduktion auf ca. 13 aussagekräftige Koeffizienten. MFCCs basieren aber auch auf vielen Designentscheidungen zur Optimierung der Repräsentation im Sinne menschlicher Voraussetzungen. In Zusammenarbeit mit KNNs könnte dies deren eigenständige Adaptivität beschränken (vgl. Abschnitt 1.2).

2.1.2. Symbolische Repräsentationen

Musik ist ein Spezialfall von Audiodaten mit spezieller Struktur. Als Definition kann man verallgemeinern: Werke der Musik sind Kunstwerke aus organisierten Schallereignissen. Das ermöglicht eine Repräsentation solcher Werke durch symbolische Beschreibung der einzelnen Ereignisse.

Das bekannste und gängigste Werkzeug von praktizierenden Musikern ist die Notenschrift, die im einfachsten Falle mindestens die Parameter Tonhöhe und Tondauer angibt. Werden die Stücke komplexer unterfällt die Notenschrift allerdings schnell dem Fluch der Dimensionalität. Bei klassischen Orchesterwerken sind es mit Tonhöhe, Tondauer, Dynamik (Lautstärke), Tempo,



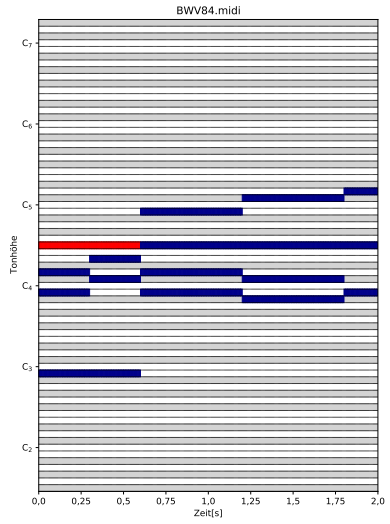
(a) handschriftlich

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE score-partwise  PUBLIC "-//Recordare//DTD
MusicXML 3.0 Partwise//EN" "http://www.musicxml.org/dtds/-
partwise.dtd">
<score-partwise version="3.0">
[...]
  <score-part id="P1">
    <part-name>Soprano</part-name>
    <part-abbreviation>S.</part-abbreviation>
  </score-part>
  <part id="P1">
    <measure>
      <attributes>
        <divisions>10080</divisions>
        <key>
          <fifths>2</fifths>
          <mode>minor</mode>
        </key>
        <time symbol="common">
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>F</step>
          <alter>1</alter>
          <octave>4</octave>
        </pitch>
        <duration>10080</duration>
        <type>quarter</type>
        <stem>up</stem>
      </note>
    </measure>
```

(b) MusicXML

```
0, 0, Header, 1, 4, 1024
1, 0, Start_track
1, 0, Tempo, 600000
1, 0, Title_t, "Soprano"
1, 0, Pitch_bend_c, 0, 8192
1, 0, Program_c, 0, 0
1, 0, Key_signature, 2, "minor"
1, 0, Time_signature, 4, 2, 24, 8
1, 0, Note_on_c, 0, 66, 90
1, 1024, Note_off_c, 0, 66, 0
1, 1024, Note_on_c, 0, 71, 90
1, 2048, Note_off_c, 0, 71, 0
1, 2048, Note_on_c, 0, 73, 90
1, 3072, Note_off_c, 0, 73, 0
1, 3072, Note_on_c, 0, 74, 90
1, 4096, Note_off_c, 0, 74, 0
1, 4096, Note_on_c, 0, 73, 90
1, 5120, Note_off_c, 0, 73, 0
1, 5120, Note_on_c, 0, 71, 90
1, 6144, Note_off_c, 0, 71, 0
1, 6144, Note_on_c, 0, 73, 90
1, 7168, Note_off_c, 0, 73, 0
1, 7168, Note_on_c, 0, 70, 90
1, 9216, Note_off_c, 0, 70, 0
1, 9216, Note_on_c, 0, 66, 90
```

(c) lesbare MIDI-Datei



(d) Pianoroll

Abbildung 2.5.: Unterschiedliche symbolische Repräsentation von J. S. Bachs „Ich bin vergnügt mit meinem Glücke“ (BWV84). Die erste Note f' der führenden Sopranstimme ist in allen vier Darstellungen rot markiert.[1]

Timbre und Artikulation schon mindestens sechs Dimensionen. Die letzteren beiden können dabei leicht in weitere Dimensionen zerlegt werden. Dieser Umstand ist gerade bei maschinellem Lernen ein zentrales Problem, da aus vergleichsweise wenigen Beispielen eine hochdimensionale Verteilung gelernt werden soll. Prokofiev [1978] identifizierte dieses Problem als kombinatorische Explosion.

Moderne Musik geht zudem meist weit über die genannten Parameter hinaus, sodass eine symbolische Beschreibung die Stücke nur noch vereinfacht abbilden kann. Ihr Vorteil gegenüber den puren Audiosignalen besteht hingegen klar in der präzisen Definition musikalischer Strukturen. Abbildung 2.5 zeigt vier symbolische Repräsentationsmöglichkeiten, die im Folgenden erläutert werden.

[1] Man beachte: Die rot markierte Note in Abbildung 2.5a ist tatsächlich ein f', da die Sopranstimme im veralteten Diskantschlüssel notiert ist.

Notenschrift

Unter Musiknoten versteht man gemeinhin die moderne westliche Notenschrift. Es sei erwähnt, dass es auch andere Notenschriften gibt, wie die Tabulatur für Saiteninstrumente oder unabhängig von der westlichen Musikpraxis historisch gewachsene wie die chinesische Jianpu-Notation oder das indische Swaralipi. Diese erreichen aufgrund anderer Zielsetzungen eine geringere Komplexität und sollen deshalb hier nicht weiter behandelt werden.

Die westliche Notenschrift zeichnet sich durch den Anspruch möglichst eindeutiger Regeln aus [vgl. Wolf, 1919] und definiert damit eine Beschreibungssprache. Abbildung 2.5a zeigt ein handschriftliches Original von J. S. Bach. Doch in moderner Popularmusik werden wie auch bereits in der Neuen Musik des 21. Jahrhunderts die ehemals akademischen Regelgerüste im Zuge einer künstlerischen Befreiung gebrochen.

MusicXML

Um das grafische Wesen der Musiknotation in eine Kodierung zu überführen, wurden verschiedene Ansätze verfolgt. Zu nennen sind beispielsweise *Notation Interchange File Format* (NIFF), *MuseData*, *ABC-Notation*, *MusiXTEX* oder *LilyPond* [Collins, 2010, S.320–322]. Vorherrschend ist mittlerweile das offene Dateiformat *MusicXML*, das wie der Name verrät Notation in der hierarchisch strukturierten Metasprache *eXtensible Markup Language* (XML) kodiert.

Die XML-Hierarchie ist (vereinfacht) wie folgt aufgebaut[2]: Eine übergeordnete Wurzel (`score`) enthält mehrere `Part`-Blätter. Diese enthalten mehrere Takte (`measure`), die als Attribute Notenschlüssel, Taktart und Tonart und als Objekte sukzessiv die Noten (`note`) mit jeweils festgelegter Tonhöhe (`pitch`) und Notenwert (`duration`) enthalten können. Ein Beispiel ist in Abbildung 2.5b gegeben.

Die hierarchische Struktur bietet eine simple konsistente Kodierung der Notationselemente. Die Überführung in einen serialisierbaren Eingabevektor für ein maschinelles Lernverfahren ist aber problematisch. MusicXML hat zudem oftmals einen großen Overhead durch viele Metainformationen und die speicherintensive redundante XML-Syntax.

MIDI

Moderne computergestützte Musik hat ihren Anfang in den frühen 1980ern. Das *Musical Instrument Digital Interface* (MIDI [MMA, 1991]) wurde ursprünglich zur Ansteuerung von elektronischen Instrumenten entwickelt und ist somit vom Konzept her bereits auf die automatische Verarbeitung ausgerichtet. Die Kodierungsmetapher ist die Bedienung einer Klaviertastatur: Entweder wird eine der Tasten (identifiziert durch eine *key*-Nummer) mit einer bestimmten Anschlagsstärke (*velocity*) gedrückt (NoteOn-Event), oder es wird eine Taste losgelassen (NoteOff-Event). Darüberhinaus gibt es eine Vielzahl von Meta-Events, die Informationen über beispielsweise Instrumente, Stimmung, Taktart oder Tempo enthalten. Diese Events sollen an dieser Stelle nicht weiter behandelt werden, näheres sei der Spezifikation [MMA, 1991] entnommen.

Die Struktur eines Musikstücks in MIDI kann als Stream von Events verstanden werden. Durch Angabe eines Zeitstempels für jedes Event entsteht das sequenzierbare Dateiformat *Standard-*

[2] Details der aktuellen Spezifikation siehe: `https://www.musicxml.com/for-developers/` (zul. abgerufen am 9.6.2021)

MIDI-Files (SMF). Zur Organisation können die Events auf bis zu 16 separate Kanäle verteilt werden, wobei der Kanal 10 typischerweise dem Schlagwerk vorbehalten wird. SMF ist dabei ein reines Binärformat in Byte-Blöcken. Lesbarkeit kann beispielsweise mit dem Tool `MIDICSV`[3] erreicht werden, wie Abbildung 2.5c zeigt. Durch geschickte Komprimierung der Redundanzen können Ansätze des Maschinenlernens für Texte auf die Daten angewendet werden (mehr dazu in Abschnitt 6.1)

Die einfache Struktur von MIDI (im Vergleich zu beispielsweise MusicXML) wird durch starke Vereinfachungen ermöglicht: Neben dem Verlust von musikalischen Parametern wie Timbre und Artikulation ist auch die Auflösung in Bytes, also einer Diskretisierung aller Parameter (darunter Anschlagsstärke und auch die Zeitstempel pro Takt) auf 128 Werte, nach heutigen Maßstäben sehr gering. In modernen Musikproduktionen und Musiksoftware hat MIDI trotz seiner Einschränkungen die Aufgabe von Notenschrift im Sinne des Notentransfers quasi abgelöst.

Pianoroll

Die Idee der *Pianoroll* kommt von der mechanischen Notenrolle. Diese waren in Lochschrift kodierte Kompositionen oder auch Aufnahmen einer Klavier-Performance. Sie sind heute noch in Drehorgeln oder in den vor der Erfindung der Schallplatten sehr populären selbstspielenden Reproduktions-Klavieren zu finden.

Das Drücken einer Taste ist kodiert durch ein Loch. Durch Drehung der Rolle entstehen zeitliche Sequenzen von Lochabfolgen für alle möglichen Tasten. Eine Pianoroll-Darstellung ist demnach leicht in die Repräsentation durch einen MIDI-Stream mit NoteOn- und NoteOff-Events zu überführen und umgekehrt. Aus dem eindimensionalen Stream wird ein zweidimensionales Pianoroll-Diagramm mit den Achsen Zeit und Tonhöhe (vgl. Abbildung 2.5d).

Da MIDI über 16 einzelne Kanäle Tonerzeuger ansprechen kann, ist eine einzelne Pianoroll für komplexere Stücke schnell überfüllt, sodass sich Noten überlappen und verdecken. Die Darstellung eines einzelnen Stückes durch 16 Pianorolls ist andererseits unübersichtlich (vertikale Stapelung führt zu $16 \cdot 128 = 2048$ Zeilen). Zudem verlieren sich die harmonischen Zusammenhänge, da gleiche Tonhöhen getrennt werden. Abhilfe schafft hier nur die dreidimensionale Anordnung, die wiederum schlecht visualisierbar ist. Diese Arbeit schlägt im Praxisteil eine Teilung in harmonische Pianoroll und perkussive *Drumroll* (Midi-Kanal 10) vor (siehe Abschnitt 6.1).

2.1.3. Metamerkmale

Eine weitere Ebene der Abstraktion und damit eine Dimensionsreduktion bietet die Konstruktion von *Metamerkmalen*. Die Vorsilbe „Meta“ (griech. für „danach“, „jenseits“) bedeutet hier: Die Merkmale sind als Repräsentant für eine Mehrzahl von Untermerkmalen definiert. Ein einfaches Beispiel sind die statistische Kenngrößen für Verteilungen.

In der MIR-Forschung gibt es eine Reihe von Arbeiten zur Ableitung von Metamerkmalen, die meist semantische Informationen tragen. Diese finden sich beispielsweise im Projekt *jMIR* gebündelt [McKay, 2010]. Für Audiodaten ist die Gewinnung von höherwertigen Metamerkmalen wie Beat- und Pitch-Erkennung und Analyse, Energiehistogramme, Grundfrequenzbestimmung oder auch komplexe Modelle wie das *Linear Predictive Coding* (Modellierung des menschlichen

[3] `https://www.fourmilab.ch/webtools/midicsv/` (zul. abgerufen am 9.6.2021)

Sprachtrakts) von besonderem Interesse. Auf symbolischen Daten stehen Metamerkmale wie Intervalverhältnisse, harmonische Analysen, Analysen des Metrums und der Taktarten, Verhältnisse von harmonischen zu perkussiven Instrumenten oder allgemein *Klangtexturen* im Vordergrund, da sich musikalische Eigenschaften aus Noteninformationen sehr viel leichter herleiten lassen. Die abstraktesten Metamerkmale stellen automatische Lösungen wie Instrumentenerkennung, Genre-Klassifikation oder Musiksegmentierung dar.

Erfolgreiche Lernverfahren wie Support Vector Machines oder Random Forest arbeiten meist am effektivsten auf statistischen und semantischen Metamerkmalen, vor allem wenn ihre Auswahl an die jeweilige Anwendung angepasst wird (*Feature Selection*). KNNs arbeiten oft erfolgreicher auf den zugrundeliegenden Rohdaten. Sie besitzen die nötige Komplexität, um entscheidende Merkmale als statistische Verteilungen aus Stichproben (Trainingsbeispiele) selbstständig abzuleiten [vgl. Li u. a., 2010].

2.1.4. Synthetisierung und automatische Transkription

Im Prinzip ist der Transfer von symbolischen Daten in Audiodaten möglich. Die Generierung von Audiosignalen aus MIDI-Streams mit Klangerzeugern ist sogar Teil des konzeptionellen Entwurfs von MIDI. Es gibt gar Musikgenres die sich dieser Art von Klangerzeugung verschrieben haben wie Techno oder Electronica. Die meisten traditionelleren Genres sind aber eher unzufriedenstellend abbildbar. Neben komplex spielbaren Instrumenten wie Saxophon oder (Solo-)Violine stellt vor allem die Erzeugung künstlichen Gesangs die größte Hürde dar. Mechanische Instrumente wie das Klavier oder gar ihre elektrischen Varianten sind eher zufriedenstellend in ihrer künstlichen Nachbildung, meist durch Aufnehmen der einzelnen Töne (*Sampling*) und spätere Zusammensetzung (durch sogenannte *Sampler*).

Die umgekehrte Aufgabe der Generierung von Symbolen aus Audio ist eine eher wissenschaftliche Angelegenheit. Das sogenannte *signal-to-symbol problem* [Collins, 2010, S.35] ist bis heute unzureichend exakt bei Einbezug aller möglichen Arten von Musik. Eine noch konkretere Aufgabenstellung ist die *automatische Musiktranskription*, die Notenschrift aus Audiodaten herzustellen versucht. Sie setzt sich dabei aus verschiedensten Teilproblemen zusammen, darunter Notenanfangs-Erkennung, mehrstimmige Tonhöhenerkennung, harmonische Analysen und Instrumentenerkennungen. Diese sind als komplexe Metamerkmale bereits im Einzelnen schwer. Deshalb bleiben vollautomatische Transkriptionen bis heute eher unzufriedenstellend. So sind sie als Eingabe für maschinelle Lernverfahren eher nicht geeignet.

2.2. Architekturtypen

KNNs gibt es in verschiedenen Ausprägungen, die sich aufgrund von strukturellen Merkmalen in unterschiedliche Typen einteilen lassen. Dieser Abschnitt stellt die drei populärsten Ansätze vor, die dann im praktischen Teil auch Anwendung finden. Die Erläuterungen beziehen sich dabei im Rahmen dieser Arbeit stets auf Musikdaten. Für allgemeine und weiterführende mathematische Erklärungen wird auf Goodfellow u. a. [2016] verwiesen.

Abbildung 2.6 zeigt Arbeiten zu KNNs im Kontext von Musik in chronologischer Einordnung. Man erkennt gewisse Meilensteine, bei denen sowohl Repräsentationen (Abbildung 2.6,

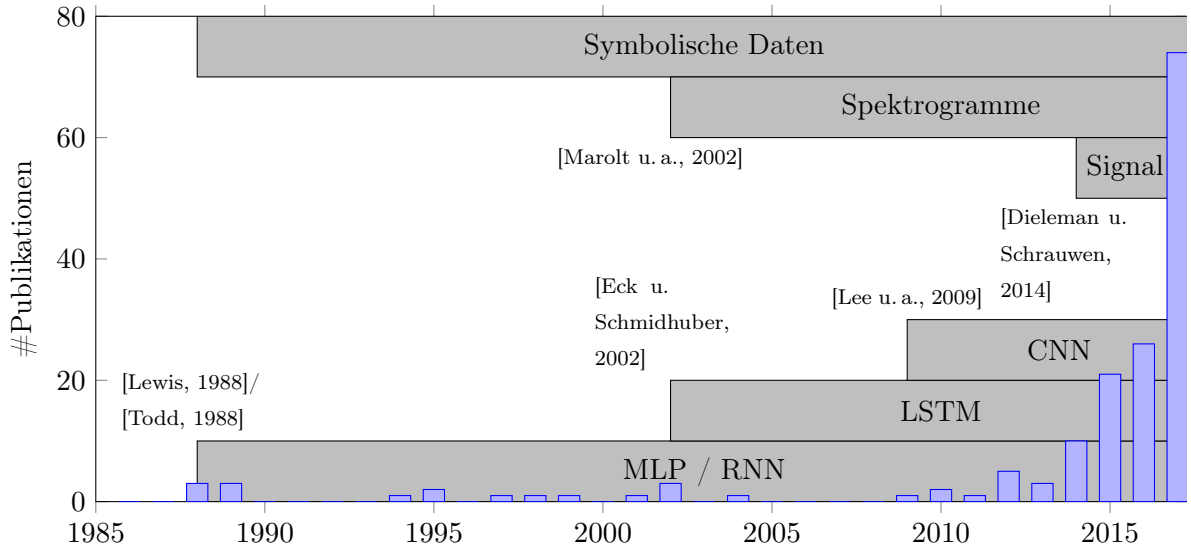


Abbildung 2.6.: Meilensteine des *DeepLearning* auf Musikdaten.[4]

oben) aber auch Strukturtypen von KNNs (ebd., unten) zum ersten Mal angewendet wurden. Wenn man die unverarbeitete Repräsentation des Audiosignal als Basis der anderen identifiziert (vgl. Abschnitt 2.1), fanden die unterschiedlichen Eingaberepräsentationen in umgekehrter Reihenfolge ihrer Abstraktion Einzug. Dies erklärt sich dadurch, dass die puren Informationsträger anfangs zu komplex waren, um in den Systemen verarbeitet zu werden. Heute können die Systeme mit der Fülle an Daten umgehen und werden aufgrund des damit ebenso steigenden Informationsgehaltes erfolgreicher. Werden die Daten vollständig unverarbeitet in ein KNN eingegeben, spricht man von einem *end-to-end*-System.

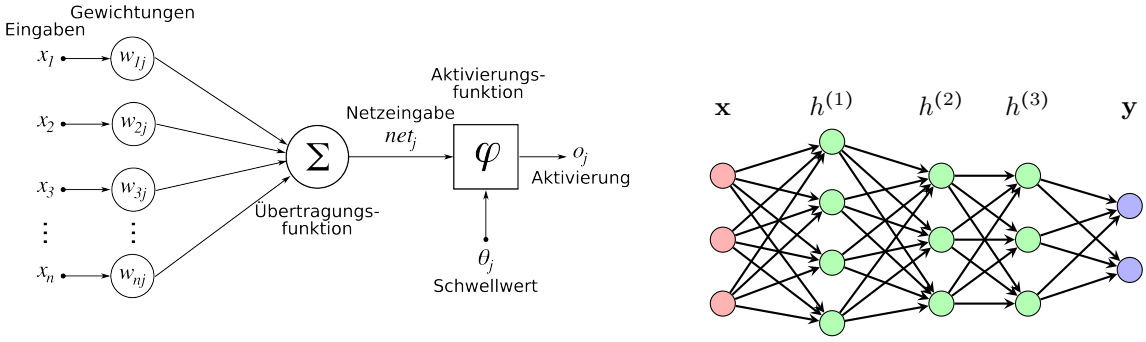
Die Architekturtypen fanden ebenso in der Reihenfolge ihrer strukturellen Komplexität Einzug. Die Ansätze werden in den folgenden Abschnitten 2.2.1 bis 2.2.4 in chronologischer Reihenfolge eingeführt.

2.2.1. Mehrlagiges Perzeptron

Das *mehrlagige Perzeptron* (MLP) ist das grundlegende Modell des modernen tiefen Lernens (engl. *Deep Learning*). Ziel ist das Approximieren einer zu lernenden Funktion $\hat{y} = f^*(\mathbf{x})$, die bei Eingabe $\mathbf{x}$ eine Kategorie $\hat{y}$ ausgibt (Klassifikation). Das MLP ist dabei eine approximierende Funktion $\mathbf{y} = f(\mathbf{x})$, die Wahrscheinlichkeiten für eine begrenzte Menge möglicher Klassen durch $\mathbf{y}$ ausgibt. [vgl. Goodfellow u. a., 2016, Kapitel 6]

Die Funktion f ist dabei eine zusammengesetzte Funktion nach biologischem Vorbild des neuronalen Netzes (vgl. Abschnitt 1.3). Die einzelnen Schalter werden als *künstliche Neuronen* bezeichnet (siehe Abbildung 2.7a). Ein Neuron j überträgt eine Menge an Eingaben x_i mit $i \in \{1, 2, \ldots, n\}$ in die sogenannte Aktivierung o_j. Die einzelnen Eingaben werden mit Gewichten w_ij multipliziert und bilden gemeinsam die Netzeingabe net_j. Die Aktivierungsfunktion ϕ bestimmt dabei, ab welchem Schwellwert θ_j eine Aktivierung erfolgt. Dabei ist die Wahl von ϕ wichtig für den Erfolg eines KNNs. Der De-facto-Standard in modernen Systemen ist die

[4] Basiert auf den Daten aus *DL4M* (Deep Learning for Music), siehe: `https://github.com/ybayle/awesome-deep-learning-music/` (zul. abgerufen am 9.6.2021)



(a) Künstliches Neuron der Schicht j.

(b) Vollständige Verknüpfung von Schichten.

Abbildung 2.7.: Innere und umliegende Struktur künstlicher Neuronen.
[Grafik links aus Wikimedia, 2005]

Rectifier-Funktion $r(z) = max\{0, z\}$ [vgl. Goodfellow u. a., 2016, S.174f]. Ihre annähernde Linearität vereinfacht die Optimierung durch Gradientenverfahren (näheres folgt in Abschnitt 2.3).

Bei der einfachsten Anordnung mehrerer Neuronen wird jedes Neuron mit dem gesamten Eingabevektor $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ verschaltet (*fully-connected*). Da alle Neuronen dieselbe Eingabe bekommen, spricht man in ihrer Gesamtheit von einer Neuronenschicht. Bei nur einer Schicht nennt sich die Architektur *einlagiges Perzeptron*. Dieses hat zunächst dieselbe klassifikatorische Macht wie ein linearer Diskriminator der Form $y_i = \sum_{i=1}^{n} w_i x_i$ (vgl. Übertragungsfunktion Σ in Abbildung 2.7a).

Das „mehrlagig“ in MLP bedeutet, die Struktur wird auf n solche Schichten erweitert (siehe Abbildung 2.7b). Bei $n = 3$ beispielsweise entsteht die verkettete Funktion $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. Die Länge der Kette wird als *Tiefe* des Netzes bezeichnet. Die Ausgaben der Funktionen $f^{(k)}$ haben dabei keine eigenständige Aussagekraft und werden deshalb als verdeckte Schichten (engl. *hidden layers*) bezeichnet.

2.2.2. Rekurrentes Neuronales Netzwerk

Ein MLP wird *feedforward*-Netz genannt, da alle Verknüpfungen in eine Richtung evaluiert werden. Die Idee liegt nahe, Ausgaben von Schichten und somit Informationen auch an Neuronen derselben oder vorangegangener Schichten zurückzuführen. Solch eine Konstruktion nennt sich *rekurrentes neuronales Netzwerk* (RNN) [Goodfellow u. a., 2016, Kapitel 10]. Rekurrente Verschaltungen sind dem biologischen Vorbild näher als einfache vorwärtsgerichtete Netzwerke. Ihre Berechnungskosten vor allem beim Training der Netze sind allerdings ungleich höher.

Abbildung 2.8 zeigt ein vollständig vernetztes RNN. Der zu klassifizierende Eingabevektor $\mathbf{x}$ wird als Eingabe an die verdeckte Schicht h gegeben und mit der Gewichtungsmatrix $\mathbf{U}$ multipliziert. Die Ausgaben aller Neuronen der Schicht h bilden zum einen die mit $\mathbf{V}$ gewichtete Ausgabe o, werden zum anderen aber auch mit $\mathbf{W}$ gewichtet an den Eingang von h zurückgeführt. Diese Rekurrenz führt zu einem sequentiellen Verhalten des Netzes bei sequenzieller Eingabe von $\mathbf{x}$. Abbildung 2.8 zeigt rechts diese Funktionsweise in linearisierter Darstellung durch Ausrollen der Schleife um h (engl. *Unfold*-Prinzip). Das was im MLP als Schicht identifiziert wurde, ist im RNN der Rekurrenzschritt t. Hervorzuheben ist, dass jeder Eintrag im Eingabevektor $\mathbf{x}$ immer wieder von der gleichen Struktur h und vor allem den gleichen Gewichtungen $\mathbf{U}$, $\mathbf{W}$ und $\mathbf{V}$

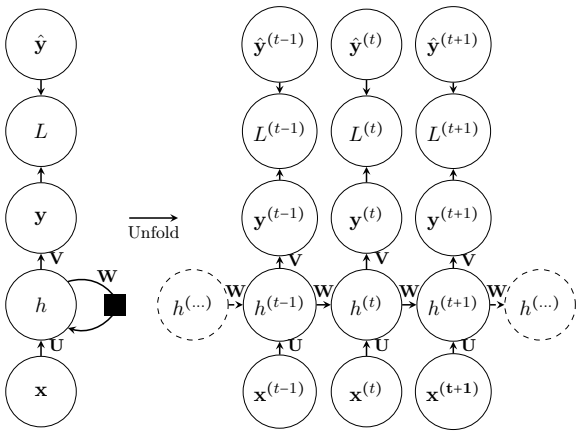


Abbildung 2.8.: Vollvernetztes RNN mit einer einzigen verdeckten Schicht h. Das schwarze Quadrat bedeutet eine Verzögerung um einen Schaltungsschritt.

verarbeitet wird. Gelernte Gewichtungen werden geteilt, um Zeitpunkt-unabhängige allgemeingültige Muster zu identifizieren. Das heißt, die Position einer Information im Eingabevektor wird irrelevant für deren korrekte Erkennung. Dieses Prinzip nennt sich *parameter sharing* und ist ein entscheidender Vorteil gegenüber MLPs, die Informationen aufgrund ihrer festen Struktur immer an derselben Position in $\mathbf{x}$ vermuten können. RNNs eignen sich deshalb besonders gut zur Analyse zeitlich geordneter und vor allem periodischer Daten wie zum Beispiel Sensorlogdaten oder Texte. Musik als Ordnung von Schallereignissen in der Zeit ist demnach ein ebenso sinnvolles Anwendungsgebiet.

2.2.3. Long Short-Term Memory

Long Short-Term Memory (LSTM) wurden zur Effizienzsteigerung von RNNs beim Erlernen von Informationen über lange Zeitintervalle entworfen [Hochreiter u. Schmidhuber, 1997]. Der Aufbau einer sogenannten LSTM-Zelle ist in Abbildung 2.9 dargestellt. Man erkennt erneut rekurrente Verbindungen, die Einfluss auf den sogenannten inneren *state* der Zelle nehmen. Im Normalfall wird der *state* auf sich selbst kopiert (*self-loop*), durch den Eingabevektor $\mathbf{x}$ (*input*) manipuliert und ausgegeben (*output*). Das LSTM erweitert die Struktur um drei sogenannte *gates*, die bei Eingabe von $\mathbf{x}$ und dem letzten *state* den Datenfluss manipulieren. So entscheidet das *input gate*, ob die aktuelle Eingabe verworfen werden soll. Das *forget gate* verwirft den Kopiervorgang des *self-loops*. Das *output gate* verwirft die Ausgabe des aktuellen *states*. Das „Verwerfen“ bedeutet mathematisch die Multiplikation mit 0 (beziehungsweise einem sehr kleinen Wert).

Der *state* kann so durch viele Iterationen stabil gehalten werden, was eine Konstanz über längere Zeiträume modelliert. Informationen können zudem langfristig akkumuliert werden, sodass Sicherheiten für einzelne Kategorien gleichmäßig wachsen. Darüberhinaus können erkannte und zur Klassifikation genutzte Merkmale bei Bedarf verworfen werden (quasi eine Speicherfreigabe), die in klassischen RNNs ansonsten zu einer Überrepräsentation geführt hätten. [vgl. Goodfellow u. a., 2016, S.409]

Das gesamte Vorgehen kann (und muss für eine Implementation) weit formaler definiert werden. Da die mathematischen Formeln im praktischen Teil nicht benötigt werden, wird an dieser Stelle darauf verzichtet und stattdessen auf Goodfellow u. a. [2016, S.410f] verwiesen.

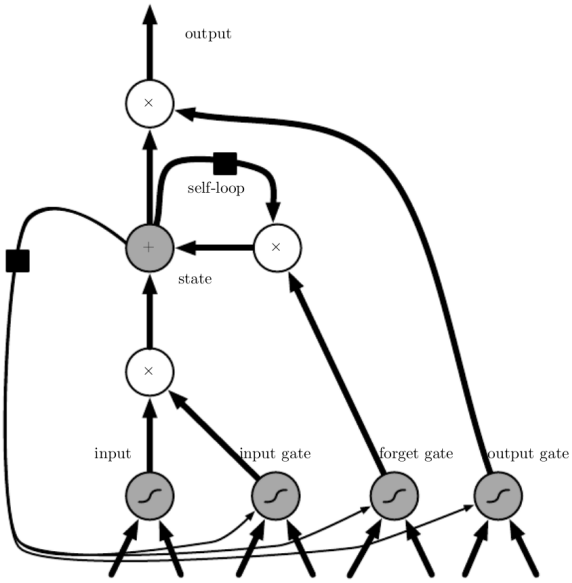


Abbildung 2.9.: Schematische Darstellung eines LSTM-Moduls. $\mathcal{S}$ steht für die Aktivierungsfunktion Sigmoid, × kennzeichnet eine Faltung.
[Grafik aus Goodfellow u. a., 2016, Abb.10.16]

2.2.4. Convolutional Neuronal Network

Lange Zeit war sich die Forschung einig, dass reine *feedforward*-Netze nicht mit den rekurrenten Varianten mithalten könnten. Das Argument war, dass die lineare Verkettung von linearen Modellen wieder nur ein lineares Modell erzeugen kann [vgl. Goodfellow u. a., 2016, S.198]. Durch le Cun [1989] wurde der äußerst erfolgreiche Typus des *Convolutional Neuronal Networks* (CNN, deut. manchmal *Faltungsnetz*) bekannt. Dieser ermöglichte vor allem eine praxistaugliche Berechenbarkeit von sehr tiefen Netzen mit einer Vielzahl an Schichten. Dies machte CNNs zu mächtigen Klassifikatoren, die die RNNs in ihrem Erfolg im Allgemeinen überholt haben.

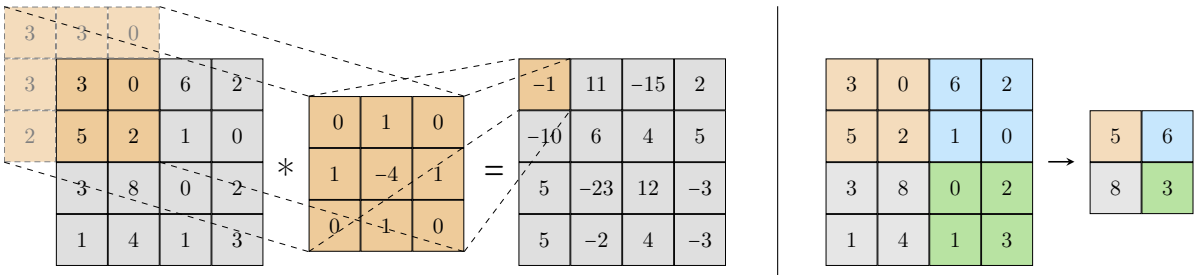
Wie der Name verrät, nutzt ein CNN die Faltung von Funktionen. Im diskreten eindimensionalen Fall ist sie definiert als

$$s(a) = (x * w)(a) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{2.4}$$

Die Faltung mit w unter der Summe übernimmt dabei dieselbe Funktion wie die gewichtete Übertragungsfunktion aus Abbildung 2.7a. CNNs werden typischerweise auf mindestens zweidimensionale Rasterrepräsentationen angewendet. Die diskrete Faltung ist dann definiert als

$$S(i,j) = (I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n-\infty}^{\infty} I(m,n)K(i-m, j-n). \tag{2.5}$$

Die gefalteten Funktionen werden als Eingabe I (*input*) und als *Kernel* K bezeichnet. Da Repräsentationen in einem digitalen Speicher immer von endlicher Größe sind, werden die Einträge außerhalb der Eingabegröße durch Nullen ersetzt und wie in der schematischen Darstellung in Abbildung 2.10a durch Wiederholen der Randwerte. Der Kernel als gelernte Gewichtungsregel übernimmt die Aufgabe benachbarte Eingabepunkte in einen gemeinsamen Merkmalswert



(a) 2D-Faltung (b) Max-Pooling

Abbildung 2.10.: Die wichtigsten Funktionen eines CNN.

zu überführen. Dabei können mehrere Kernel als unabhängige Filter die Eingabe falten und so einen dreidimensionalen Merkmalsraum erzeugen. Da die Ausprägungen der einzelnen Kernel während der Faltung konstant sind, ist das Kernel-Prinzip ebenso dem *parameter sharing* zuzuordnen. In der Praxis bedeutet dies, dass die Position eines Objektes (z.B. in einem Bild) zur Merkmalsdetektion durch Kernelfaltung zunächst irrelevant ist.

Ein weiteres Prinzip, dass die Verarbeitung von tiefen Netzen beschleunigt, ist die Erhöhung der Schrittgröße der Faltung. Gleichung 2.5 nutzt aufgrund der ganzzahligen Diskretisierung implizit eine Schrittgröße $l = 1$. Eine Schrittgröße von $l = 2$ würde das Auslassen jedes zweiten Wertes bedeuten. Der Merkmalsraum verkleinert sich dabei um den Faktor $\frac{1}{l}$. Teilen die benachbarten Eingabewerte Informationen, so bedeutet ihre Zusammenfassung im besten Falle die Abstraktion zu einem sinnvollen Merkmal.

In modernen Netzarchitekturen wird diese Aufgabe häufig durch eine sogenannte *Pooling*-Schicht übernommen. Beim *max pooling* wird ein meist quadratischer Filter angewendet, der schrittweise über den Merkmalsraum gelegt wird und an jeder Position den Wert der maximalen Neuronenaktivierung übernimmt (vgl. Abbildung 2.10b). Obwohl dieser Filter nicht gelernt ist, zeigt seine Anwendung fast ausnahmslos signifikante Vorteile für die korrekte Klassifikation. Das erklärt sich bisher nur mutmaßlich mit der Verminderung des Speicherbedarfs und der damit einhergehenden schnelleren Berechenbarkeit, was wiederum deutlich tiefere Netze möglich macht und so das Lösen komplexerer Aufgaben.

Da die Informationsgewinnung eines CNNs von der semantischen Sinnhaftigkeit der Kernel abhängt, arbeiten diese besser, wenn mehr Dimensionen verarbeitet werden können. So ist die Domäne Bild mit dem zweidimensionalem Pixelraster und der Nachbarschaftsbeziehungen der Bildinformationen sehr erfolgreich. Trennt man die Farbkanäle auf, entsteht ein dreidimensionaler Eingabevektor und der Erfolg steigt prinzipiell weiter. Wie in Abschnitt 2.1 dargelegt, bestehen Audio- und Musikdaten aus eindimensionalen sequentiellen Strukturen. Die Ansätze einer Transformation in mehrdimensionale Spektralbilder und Notendiagramme (vgl. Abschnitte 2.1.1 und 2.1.2) erscheinen für CNNs zunächst sehr gewinnbringend. Aktuelle Arbeiten weisen aber darauf hin, dass diese Vermutung unter Umständen nicht allgemeingültig ist und weiterer Untersuchung bedarf. So finden sich auch für eindimensionale Kernel mittlerweile ebenso erfolgreiche Anwendungen (z.B. in der Sprachsynthese [van den Oord u. a., 2016]).

Den größten Erfolg haben CNNs im Bereich der Bildverarbeitung. Dies liegt mutmaßlich auch an der Dominanz dieser Domäne (vgl. Abschnitt 1.3). Auch die Wirtschaft zeigt heute großes Interesse am Forschungsbereich, vor allem nachdem Krizhevsky u. a. [2012] den populä-

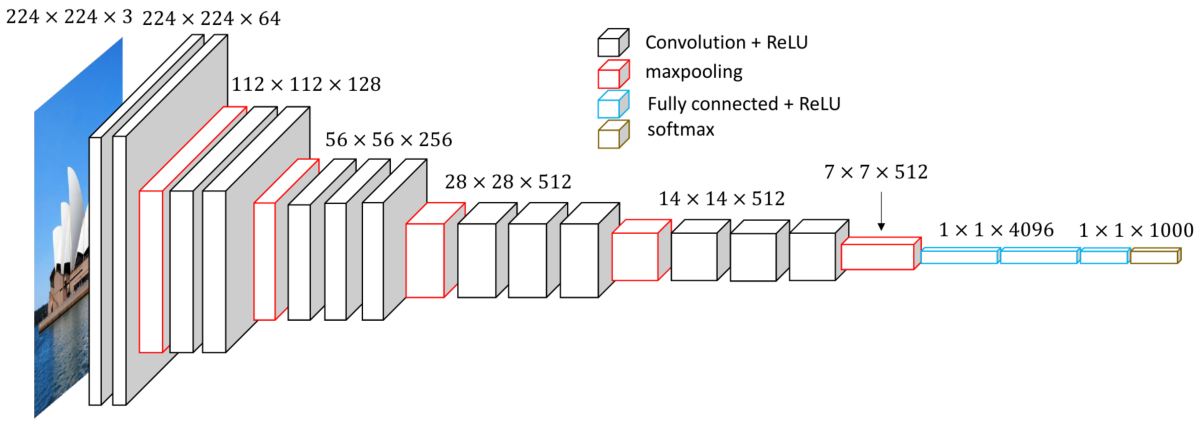


Abbildung 2.11.: Schematik der CNN-Architektur des etablierten VGG16-Modells.
[Grafik aus Wang u. a., 2017, Abb.2]

ren ImageNet-Wettbewerb gewinnen konnte. So haben CNNs andere Methoden bei derartigen Wettbewerben mittlerweile weit überholt. Dies war die vorläufige Spitze einer sich langjährig anbahnenden Entwicklung [vgl. Goodfellow u. a., 2016, S.371f]. Heutige Anwendungen auf Musikdaten zeigen eine potentiell ähnliche Entwicklung, wie die frühe Arbeit von Lee u. a. [2009] zeigt. Merkmalsgewinnung (selbst mit *end-to-end*-Methodik) wurde hier erfolgreich angewendet [Dieleman u. Schrauwen, 2014; Li u. a., 2010].

2.2.5. Modellierung in Schichten

Die Strukturtypen MLP, RNN und CNN getrennt zu betrachten ist aufgrund ihrer Komplexität im Einzelnen sinnvoll. In der Praxis treten die Strukturen aber meist gemischt auf. Das Prinzip der Schichten aus Abschnitt 2.2.1 wird dazu zu einem zentralen Konzept erhoben.

RNNs und CNNs haben aufgrund des *parameter sharings* den Vorteil der positionsunabhängigen Erkennung. Die MLPs wachsen zwar exponentiell mit der Anzahl ihrer Neuronen, zeichnen sich aber gerade durch die vollständige Vernetzung als unabhängige Verknüpfung aller einzelnen Informationen aus. In der Praxis werden deshalb oft erstgenannte Ansätze vor der Ausgabe mit vollvernetzten Schichten (engl. *fully connected layer*, auch *dense layer*) ergänzt. So entstehen tiefe Netze mit unterschiedlichen Schichten, wie beispielsweise das sehr tiefe und äußerst erfolgreiche CNN aus Simonyan u. Zisserman [2015] (siehe Abbildung 2.11). Aber auch die Verschaltung von CNNs und RNNs kann Vorteile haben. So zeigen Vinyals u. a. [2015] wie ein CNN aus einem Bild Merkmale extrahiert, mit deren Eingabe ein LSTM sinnvolle Beschreibungssätze generieren kann.

Im Praxisteil dieser Arbeit werden folgende Schichten unter den jeweils fettgedruckten Bezeichnungen genutzt:

- **Dense**, vollvernetzte „dichte“ Schicht (*fully-connected*) wie beim MLP
- **LSTM**, ein Modul ist eine Schicht
- **CNN** oder **Conv**, eine einzelne Faltungsschicht kann dabei mehrere Kernel haben
- **MaxPool**, zusammenfassende Pooling-Schicht (siehe Abschnitt 2.2.4)

- **Dropout**, regularisierende Schicht (Erklärung folgt in Abschnitt 2.3.3)
- **Flatten**, übersetzt ein mehrdimensionales Eingangsmatrix zeilenweise in einen Vektor

2.3. Training

Nach der Einführung der Strukturen muss nun erklärt werden, wie die Netzwerke trainiert werden können. Training bedeutet die Gewichte $\mathbf{w}$ eines Netzes so zu wählen, dass die zu approximierende Funktion $y = f^*(\mathbf{x})$ durch das Netz, welches die approximierende Funktion $\hat{\mathbf{y}} = f(\mathbf{x})$ modelliert, möglichst exakt abgebildet wird. Das Training ist also ein Optimierungsproblem. Lässt sich die Abweichung von einer Ausgabe vom eigentlich gewünschten Wert durch die quadratische Fehlerfunktion

$$E(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2 \tag{2.6}$$

berechnen, so können die Gewichte eines KNNs mittels der Delta-Regel schrittweise optimiert werden:

$$\Delta w_{ji} = \eta(\hat{y}_j - y_j) \tag{2.7}$$

Der Klammerterm stellt dabei die Ableitung von E dar. Die Delta-Regel ist somit ein Gradientenverfahren. Die Trainingsbeispiele werden nacheinander wiederholt zugeführt. Ein kompletter Zyklus wird *Epoche* genannt. Das i-te Gewicht des j-ten Neurons wird mit jedem Beispiel um Δw_{ji} angepasst. Der Faktor $\eta \in [0, 1]$ nennt sich *Lernrate* und variiert die Stärke der Gewichtsanpassung. Vor Beginn des Trainings werden die Gewichte typischerweise entweder mit 0 oder zufällig initialisiert. Da die erwartete Ausgabe y zum Training bekannt sein muss, gehört der Algorithmus zur Gruppe der *überwachten Lernverfahren*.

Für eine Klassifikation muss zudem die Fehlerfunktion E aus Gleichung (2.6) ersetzt werden. Für ein Multiklassen-Problem ist die kategorische Kreuzentropie weit verbreitet, wobei die korrekte Ausgabe y als *One-Hot*-Vektor[5] $\mathbf{y}$ kodiert wird:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i}^{N} \hat{\mathbf{y}}_i \log(\mathbf{y}_i) \tag{2.8}$$

L ist beim Training das negative Bewertungsmaß. Man spricht von der sogenannten Verlustfunktion (*Loss*).

2.3.1. Erweiterungen für tiefe Netze

Die Delta-Regel ist zunächst nur auf einlagige Netze anwendbar. Zur Anwendung auf *feedforward*-Netze mit mehreren Schichten wird die Delta-Regel zur sogenannten *Backpropagation*-Methode (deut. manchmal *Fehlerrückführung*) erweitert. Diese nutzt die Kettenregel um die Gradienten für die Neuronen der verdeckten Schichten abzuleiten. So werden die Gewichtsanpassungen ausgehend vom Ausgabefehler L zur Eingabe des Netzes zurückberechnet. Die Ausgabeschicht wird dabei mit der Delta-Regel optimiert. Die verdeckten Schichten beziehen rekursiv die Fehler aller nachfolgenden Neuronen mit ein. [Goodfellow u. a., 2016, S.204ff]

[5] Ein *One-Hot*-kodierter Vektor hat einen Wert von 1 an der Stelle i der entsprechenden Klasse und sonst Nullen.

Ein besonders bei RNNs auftretendes Problem nennt sich *verschwindender Gradient*. Vor allem gewollt hohe Gewichtungen nahe 1 schwächen sich bei wiederholter Multiplikation exponentiell ab (z.B. ist $0{,}99^{100} = 0{,}37$). Das erschwert die Anwendung von Backpropagation auf RNNs, vor allem wenn das Verknüpfen entfernter Informationen gefragt ist. Ein LSTM kann durch das Konzept des inneren *states* diesem Problem entgegenwirken.

Ein weiteres Problem, das alle Gradientenverfahren teilen, ist ihre Unfähigkeit ein lokales Extremum vom Globalen zu unterscheiden. Nähert sich ein einfaches minimierendes Gradientenverfahren wie die Delta-Regel einem lokalen „Tal“, wird es unter Umständen nie wieder herauslaufen. Verbesserung schafft die Einführung eines Trägheitsterms:

$$\Delta w_{ji}(t+1) := \alpha \Delta w_{ji}(t) - \eta(\hat{y}_j - y_j), \tag{2.9}$$

Die Gewichtsanpassung Δw_{ji} im Schritt t+1 hängt hier von der Anpassung im vorherigen Schritt t ab. Als Analogon zur Massenträgheit in der Physik möchte Δw_{ji} seine aktuelle Geschwindigkeit beibehalten und kann so gegebenenfalls über das Tal eines lokalen Minimums „hinausschießen“. Der Faktor $\alpha \in [0, 1]$ steuert dabei quasi die physikalische Masse und damit die Stärke dieser Trägheit.

Ein weiterer Stellwert ist die Lernrate η. Wird η zu klein gewählt, behindert dies erneut die Suche nach globalen Extrema. Zu große Lernraten hingegen optimieren eine hinnehmbar gute lokale Extremstelle nicht hinreichend. Adaptive Lernraten, die sich Schritt für Schritt verkleinern, können durch Methoden wie *AdaGrad* oder *RMSProp* gesteuert werden. Das in dieser Arbeit genutzt Optimierverfahren *Adam* [Kingma u. Ba, 2015] ist eine Erweiterung und kombiniert das Konzept von Trägheit und adaptiver Lernrate.

Typischerweise werden die Anpassungen aus dem Mittel mehrerer Trainingsbeispiele bestimmt. Dieses Vorgehen optimiert den Abstieg in Richtung des stochastisch besten Gradienten. Wird der Gradient aus dem Mittel aller Trainingsbeispiele gewählt, steigert sich der Berechnungsaufwand überlinear. Aufteilung in mehrere Teilmengen, sogenannte *Batches*, ermöglicht die Hardwareparallelisierung und reduziert damit die Laufzeit erheblich. Zudem wird bei geringerer Batchgröße weniger flüchtiger Datenspeicher zur Laufzeit benötigt. Sehr kleine Batches führen aufgrund der hohen Varianz der entstehenden Gradienten zu einer zusätzlichen Regularisierung. So muss der im Anwendungsfall beste Mittelweg zwischen effektiverem Gradienten (größere Batches) und schnellerem, möglicherweise allgemeingültigem Lernerfolg (kleinere Batches) gesucht werden. In jedem Fall sollte die Zusammenstellung der Batches nach jeder Epoche zufällig umsortiert werden, um systematische Fehler auszuschließen.

2.3.2. Bewertungsmaße

Die vorgestellten Trainingsverfahren aus dem vorherigen Abschnitt gibt es in unzähligen Varianten [vgl. Goodfellow u. a., 2016, Kap.8]. Mit ihnen ist es theoretisch möglich jede beliebige Funktion $y = f^*(\mathbf{x})$ zu approximieren. Grundlage dafür bietet eine endliche Menge $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\} \subset \{(\mathbf{x}, y) \mid f^*(x) = y\}$ an n bekannten Beispielen. Aus dieser sogenannten *Trainingsmenge* soll auf die gewünschte Obermenge geschlossen werden. Je größer n ist, das heißt je mehr Beispiele zur Verfügung stehen, desto besser stehen die Chancen die zugrundeliegende Verteilung weitestgehend korrekt abzubilden.

Bei einem automatisch gelernten Algorithmus kann die Korrektheit durch eine unabhängige *Testmenge* statistisch überprüft werden. Eine Testmenge besteht ebenso aus bekannten Beispielen, die jedoch allesamt nicht in der Trainingsmenge vertreten und dem Algorithmus somit unbekannt sind. Gilt für ein Testbeispiel $(\mathbf{x}, y)$, dass $y = \hat{y} = f(\mathbf{x})$, so ist die Ausgabe richtig (r), ansonsten ist sie falsch (f). Da y meist durch einen Vektor $\mathbf{y}$ als die Aktivierung der Ausgabeneuronen und somit als Vektor von Wahrscheinlichkeitswerten darstellt wird, ist die Vorhersage typischerweise als die Klasse mit dem höchsten Wert y_i definiert.

Ein Multiklassenproblem kann pro Klasse als binäre Klassifikation uminterpretiert werden. Dazu wird eine der Klassen als die gewünschte *positive* Klasse p und die restlichen als eine *negative* (ungewünschte) Fehlerklasse n aufgefasst. Zusammen mit den jeweiligen Häufigkeiten von r und f entstehen vier Kombinationsmöglichkeiten:

- r_p, richtig positive Klassifikation (positive Klasse lag vor)
- r_n, richtig negative Klassifikation (negative Klasse lag vor)
- f_p, falsch positive Klassifikation (negative Klasse lag vor)
- f_n, falsch negative Klassifikation (positive Klasse lag vor)

Das einfachste Maß für die Korrektheit der Klassifikation im Hinblick auf beide Klassen bildet die Korrektklassifikationsrate (engl. *accuracy*):

$$\mathrm{KKR} = \frac{r_p + r_n}{r_p + f_p + r_n + f_n}, \tag{2.10}$$

die angibt, wie viele der Testbeispiele richtig identifiziert wurden.

Ein differenzierteres Bild der Korrektheit im Hinblick auf die positive Klasse bietet die Sensitivität (auch Trefferquote, engl. *recall*):

$$R = \frac{r_p}{r_p + f_n} \tag{2.11}$$

Sie gibt an mit welcher Wahrscheinlichkeit ein positives Beispiel auch als positiv identifiziert wird.

Der positive prädiktive Wert (Genauigkeit, engl. *precision*)

$$P = \frac{r_p}{r_p + f_p} \tag{2.12}$$

gibt hingegen an, mit welcher Wahrscheinlichkeit ein als positiv erkanntes Beispiel auch tatsächlich ein positives ist, das heißt wie sicher die Erkennung für die beobachtete Klasse ist. Die Fehler und die Maße können zur Verdeutlichung der Zusammenhänge in eine sogenannte Konfusionsmatrix eingeordnet werden, wie sie Tabelle 2.1 zeigt.

Für ein Multiklassenproblem fällt für jede Klasse eine eigene Konfusionsmatrix an, aus denen wiederum die oben angegebenen und auch weitere aus der Matrix abzuleitende Fehlermaße angegeben werden können. Um Aussagen abzuleiten, werden die Informationen gebündelt. Dazu wird häufig das F-Maß als das (meist mit 1-gewichtete) harmonische Mittel über die Maße P und R genutzt.

		Wirklichkeit		
		p	n	
Vorhersage	p	r_p	f_p	P
	n	f_n	r_n	$\overline{P}$
		R	$\overline{R}$	KKR

Tabelle 2.1.: Konfusionsmatrix mit den Bewertungsmaßen KKR, P und R sowie deren Komplementen bei Umdeutung der Fehlerklasse als $\overline{P}$ und $\overline{R}$.

Da diese Arbeit ein Zweiklassenproblem behandelt, in dem beide Klassen zunächst in selber Form von Interesse sind, wird im Folgenden zunächst hauptsächlich das Bewertungsmaß KKR verwendet. Für das Vorhaben dieser Arbeit ist P von besonderer Bedeutung. Dies wird in den Kapiteln 6 und 8 näher erläutert.

2.3.3. Überanpassung

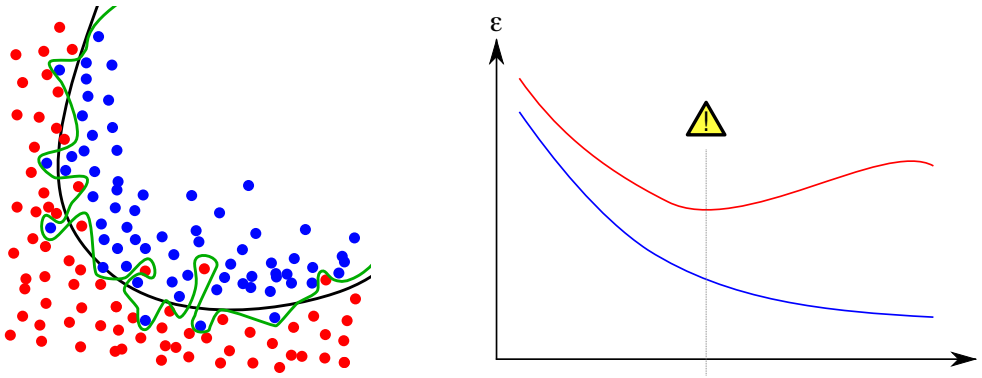
Alle Trainingsverfahren sind zunächst darauf angelegt, die Trainingsmenge möglichst vollständig und meist auch möglichst schnell zu lernen. Eine komplexe Verteilung der Trainingsbeispiele benötigt dabei auch ein komplexes beziehungsweise tiefes KNN. Spätestens wenn alle Trainingsbeispiele korrekt reproduziert werden können, tritt eine Stagnation ein, da die Verlustfunktion Werte nahe 0 ausgibt und so die Gewichtsanpassungen durch Δw_{ji} verschwinden.

Ob die tatsächliche statistische Verteilung, die die zu approximierende Funktion f^* generiert, vom KNN korrekt abgebildet wird, ist jedoch nicht sicher. Ob die Abbildung hinreichend exakt ist, entscheidet die statistische Evaluierung auf den Testdaten im Kontext des jeweiligen Anwendungsfalls. Es besteht jedoch die Gefahr, dass obwohl die Trainingsbeispiele vom Modell immer fehlerfreier abgebildet werden, eine sogenannte Überanpassung (engl. *Overfitting*) eintritt. Abbildung 2.12a zeigt solch eine überangepasste Funktion (grüne Kurve). Alle Trainingspunkte werden perfekt eingefasst. Trotzdem ist die Verteilung unnötig komplex. Obwohl die schwarze Kurve geringfügige Fehler auf der Trainingsmenge macht, kann sie als Verallgemeinerung, das heißt beim Testen auf unbekannten Daten, besser sein.

Diese Vermutung bestätigt sich nahezu ausnahmslos beim praktischen Trainieren von KNNs. Abbildung 2.12b zeigt den Verlauf des Loss (y-Achse) von Epoche zu Epoche (x-Achse). Die blaue Kurve zeigt den stetig fallenden Loss auf der Trainingsmenge. Der Loss auf den Testdaten (rote Kurve) sollte bei einem verallgemeinerbaren Lernerfolg in selbem Maße fallen. Irgendwann gibt es einen Bruch durch die Überanpassung an die Trainingsdaten und der Loss auf den Testdaten steigt wieder. Dann stellt die approximierende Funktion f keine Verallgemeinerung auf unbekannte Daten mehr dar.

Nichtsdestotrotz stellen die Kurven in Abbildung 2.12b den gewünschten Verlauf des Trainings dar, denn zunächst zeigt sich ein verallgemeinerbarer Lernerfolg und die Trainingsdaten können auch vom Modell abgebildet werden. Der Gegensatz wäre ein Modell, das zu beschränkt ist, um überhaupt die Trainingsdaten zu approximieren. Dann spricht man von Unteranpassung [vgl. Goodfellow u. a., 2016, S.115].

Der Zeitpunkt des Bruchs stellt im Training die optimale Balance zwischen Verallgemeinerung und Überanpassung dar. Der Zeitpunkt wird in der Theorie jedoch erst nach Abschluss des



(a) Konzeptionelle Darstellung (b) Praktische Erscheinung

Abbildung 2.12.: Darstellung des Problems der Überanpassung.
[Grafik aus Wikimedia, 2007, 2008]

angesetzten Trainings sichtbar. Abhilfe schafft die Online-Evaluierung unabhängiger Beispiele nach Abschluss jeder Epoche. Der sogenannte *Early-Stopping*-Mechanismus [Prechelt, 1996] überwacht die Lernkurve und bricht das Training ab, sobald in mehreren aufeinanderfolgenden Epochen keine Verbesserung mehr erreicht werden konnte. Zu beachten ist, dass zur Durchführung eine dritte Menge an unbekannten Beispielen, die sogenannte *Validierungsmenge*, zur Verfügung gestellt werden muss. Das gesteuerte Stoppen optimiert den gelernten Algorithmus auf genau diese Beispiele. Seine allgemeingültige Korrektheit muss im Anschluss auf weiteren unabhängigen Testbeispielen verifiziert werden. Es muss also eine hinreichende Menge an Beispielen zur Verfügung stehen. Mithilfe der Online-Validierung ist auch die Optimierung weiterer Parameter wie der Anzahl der Neuronen und verdeckten Schichten sowie weitere Modifikationen der gesamten Netzarchitektur bereits während des Trainings möglich. Solche Parameteroptimierungen werden von den Parametern, die vor Beginn des Trainings festgelegt werden, durch die Bezeichnung *Hyperparameter* abgegrenzt.

Methoden, um eine Überanpassung zu vermeiden, nennen sich Methoden der *Regularisierung*. Die bereits bekannte Pooling-Schicht zählt ebenso dazu, denn der ihr inhärente Informationsverlust bewirkt, dass die Anpassung an die eigentlichen Trainingsdaten gröber ausfällt (Abbildung 2.12a, schwarze Kurve). Aus dieser Beobachtung entstand auch die Idee zum *Dropout* [Srivastava u. a., 2014]. Dabei werden Ein- beziehungsweise Ausgaben zwischen Neuronen beim Training mit einer festgelegten Ausfallrate von meist 0,1 bis 0,5 genullt. Das KNN lernt demnach die Klassifikation auf Ausschnitten der Merkmale und an unterschiedlichen Stellen im Netz, benötigt dafür aber länger bis zur hinreichenden Abdeckung. Trotz der verblüffenden Einfachheit des Ansatzes bewies sich die Methode als besonders effektiv gegen Überanpassung [Goodfellow u. a., 2016, S.258ff].

2.4. Analyse fertiger Netze

Ein KNN ist zunächst eine Blackbox. Zwar sind die verdeckten Netzgewichte und Schaltungen theoretisch beobachtbar, jedoch ist dies aufgrund der enormen Größe der Netze (meist $>10^6$, eher $>10^7$) nicht praktikabel. So kann die Korrektheit eines Modells zunächst nur quantitativ auf

Testdaten evaluiert werden, wie in Abschnitt 2.3.2 beschrieben. Über die tatsächliche Funktionsweise, also die Logik hinter der Merkmalsseparierung, kann lediglich durch Beispieleingaben und Vergleich der Ausgaben gemutmaßt werden. So dient die Angabe von guten und schlechten Beispielen in vielen Arbeiten als Indikator einer verallgemeinerbaren Methodik.

Doch es ist leicht zu zeigen, dass die verallgemeinerbare Aussagekraft von KNNs äußerst fragil ist: Bei einer sogenannten *Adversarial Attack* (deut. *feindlicher Angriff*) wird die Eingabe des Netzes so manipuliert, dass scheinbar geringe Veränderungen am Eingabevektor, die die reale Erscheinung eines Objektes minimal verändern, einen fatalen Einfluss auf Falschklassifikationen haben [Xu u. a., 2020]. Es ist damit möglich, sehr hohe Aktivierungen in beliebigen Ausgabeneuronen zu forcieren. Dies birgt deshalb auch ein großes Sicherheitsrisiko. Bei *Adversarial Attacks* wird zwischen Whitebox- und Blackbox-Attacken unterschieden [Xu u. a., 2020, S.155–159]. Erstere kennt die inneren Parameter der Netze. Letztere hat lediglich einen operativen Zugang. Gerade die gezielte Multiplikation mit minimalem Rauschen kann die KNN-Ausgabe komplett unbrauchbar machen. Eine Möglichkeit, um die Robustheit zu erhöhen, ist das absichtliche Verrauschen der Trainingsbeispiele als eine Form der Augmentierung [Goodfellow u. a., 2016, S.240f]. Da die Datensätze der vorliegenden Arbeit aus zuverlässigen Generatoren stammen, ist dies hier nicht notwendig (mehr dazu folgt in Kapitel 6). Zudem bestehen keinerlei Sicherheitsbedenken.

Aufgrund des kritischen Sicherheitsaspekts bei wachsender Verbreitung von KNN-Anwendungen wird die tiefe Analyse und Interpretierbarkeit aber immer wichtiger. So stieg in den letzten Jahren das Interesse und die Forschung an Methoden zur Erkennung und Visualisierung von inneren Prinzipien. Die folgenden Abschnitte stellen etablierte Methoden vor und geben eine erste Einschätzung über die Anwendbarkeit auf Musikdatensätze. Die praktische Anwendung folgt in Kapitel 7.

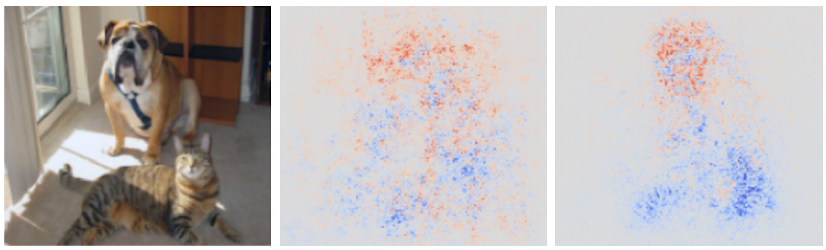
2.4.1. Attention Maps

Ziel der *Attention Maps* ist die Lokalisierung der Einträge in einem Merkmalsvektor, die den größten Einfluss auf die Klassifikationsentscheidung haben. So kann abgeschätzt werden, ob die gelernten Charakteristika die gewünschte Semantik besitzen. Bei einem Netz, dass zum Beispiel in einem Bild die Klassen „Hund“ und „Katze“ vorhersagen soll, kann geprüft werden, ob die gewünschten Objekte tatsächlich die Klassifikation bestimmt haben, oder ob andere zufällig passende Nebenobjekte (beim Beispiel mit Hund und Katze z.B. Halsband oder Wollknäuel) die Klassifikation lediglich in der Testumgebung erfolgreich erscheinen lassen.

Die ersten Arbeiten bedurften eines erneuten Trainings unter Einfügung spezieller Analyseschichten. Neue Ansätze können auf bestehende Netze angewendet werden. Sie sind deshalb sehr viel praktikabler in ihrer Anwendung. Die vom Autor als am etabliertesten identifizierten Ansätze werden im Folgenden erläutert.

Saliency Maps

Saliency Maps (deut. *Herausragen*) identifizieren die Bereiche in einer Beispieleingabe, für die die anliegenden Netzneuronen die höchste Aktivierung zeigen. Dazu werden im einfachsten Fall mithilfe des Backpropagation-Algorithmus die Gradienten von einer gewünschten Ausgabeklasse durch das Netz hindurch zu einem Eingabebeispiel zurückberechnet. Je größer der Gradient



(a) Eingabe (b) Vanilla (c) SmoothGrad

Abbildung 2.13.: Saliency Maps in Anwendung auf die Klassen „Hund“ (rot) und „Katze“ (blau). [Grafik aus Smilkov u. a., 2017, Fig.6]

für eine Stelle im Eingabevektor ausfällt, desto höher ist sein Beitrag zur Klassifikation. Dieses Vorgehen wurde zuerst von Simonyan u. a. [2014] beschrieben und wird heute als *Vanilla Saliency* bezeichnet[6].

Abbildung 2.13b zeigt die Anwendung von Vanilla Saliency auf die Klassen „Hund“ (rot) und „Katze“ (blau). Aufgrund dieser Visualisierung könnte man dem verwendeten Netz eine gewisse Ungenauigkeit zusprechen. Smilkov u. a. [2017] identifizieren jedoch ein Problem bei der direkten Visualisierung der Gradienten: Die Ableitung der Aktivierungsfunktion der Ausgabeklasse ist aufgrund der Trainingsmethodik verrauscht. Dieses globale Rauschen sind hochfrequente Schwankungen. Sie weisen nach, dass sich der Klassifikationserfolg nicht verringert, wenn das Rauschen lokal durch einen Gauß-Filter begrenzt wird. Die sich daraus ergebende Visualisierungstechnik *SmoothGrad* zeigt im Vergleich eine präzisere Lokalisierung der entscheidenden Bereiche (siehe Abbildung 2.13c).

Da alle KNNs im Grunde auf Gradientenverfahren basieren, sind *Saliency Maps* auf alle Typen anwendbar. Allerdings zeigen reine *Saliency Maps* mitunter große Unsicherheiten, vor allem bei *Adversarial Attacks* [vgl. Kindermans u. a., 2019].

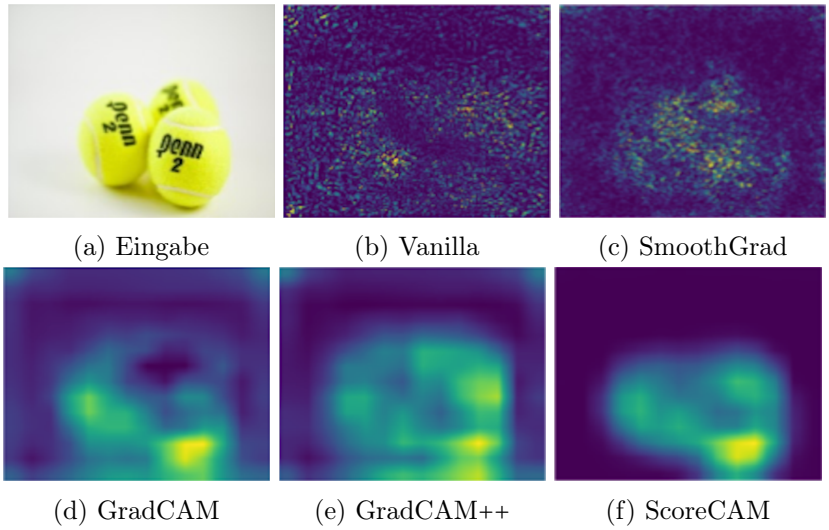
Class Activation Mapping

Das *Class Activation Mapping* (CAM) macht sich Besonderheiten der CNNs zu Nutze. Sie gehen davon aus, dass die Dense-Schichten, die typischerweise an die Faltungsschichten hinten angefügt werden, die Lokalisierung unnötig erschweren. So betrachtet das ursprüngliche CAM [Zhou u. a., 2016] ausschließlich reine Faltungsnetze.

Selvaraju u. a. [2019] optimieren diesen ersten Ansatz durch *GradCAM* (*Gradient-weighted Class Activation Mapping*). Dabei werden die Faltungsschichten aus der CNN-Architektur entfernt und die Gradienten der letzten Conv-Schicht werden als *Attention Map* interpretiert. GradCAM ist so flexibel auf diverse Architekturen anwendbar. Eine Weiterentwicklung ist *GradCAM++* [Chattopadhay u. a., 2018]. Diese optimiert das Problem, wenn die gesuchte Klasse nicht zusammenhängend oder mehrfach im Bild vertreten ist. Für tiefergehende Erläuterungen wird an dieser Stelle aufgrund der mathematischen Komplexität der Methoden auf die Literatur verwiesen.

Einer der aktuell erfolgreichsten Ansätze ist *ScoreCAM* [Wang u. a., 2020]. Dieser maskiert das Beispielbild zufällig und misst den entsprechenden *Score* der gewünschten Klasse. *Score*

[6] Der Begriff „Vanilla“ wird im englischen Sprachgebrauch analog zum deutschen „Nullachtfünfzehn“ verwendet.



(a) Eingabe (b) Vanilla (c) SmoothGrad

(d) GradCAM (e) GradCAM++ (f) ScoreCAM

Abbildung 2.14.: Einige Varianten von Attention Maps im Vergleich.
[Grafik aus Wang u. a., 2020]

bedeutet hier die Aktivierung der zugehörigen Ausgabe. Dabei werden keine Gradienten benötigt. So umgeht ScoreCAM das Problem des globalen Rauschens aus dem letzten Abschnitt. Ein Nachteil ist jedoch die enorme Rechenleistung, da das vorgeschlagene Maskieren im Monte-Carlo-Verfahren sehr oft wiederholt werden muss.

Abbildung 2.14 zeigt einen Vergleich aller bis hierher beschriebenen Methoden. So zeigt sich an der von Bild zu Bild deutlich werdenderen Trennbarkeit der drei Tennisbälle die steigende Lokalisierungspräzision der Methoden.

2.4.2. Aktivierungs-Maximierung

Die Methode der *Aktivierungs-Maximierung* untersucht die Funktion einzelner Neuronen, Schichten oder ganzer Netze, indem es Eingaben generiert, deren Aktivierung möglichst hoch ist. Dies soll eine Art perfektes Abbild der einer Klasse zugeschriebenen Merkmalsausformung erzeugen. Dazu wird der Loss einer Klasse systematisch durch ein Gradientenverfahren minimiert. Es entsteht ein artifizielles Eingabebild, das Charakteristika der zu untersuchenden Klasse visuell beschreibt. Alternativ zur Loss-Minimierung der Ausgabe können auch Aktivierungen bestimmter Schichten oder bestimmter Neuronen maximiert werden. Diese Methode lässt sich demnach für alle Arten von KNNs anwenden.

Abbildung 2.15 zeigt als Beispiel eine Klassifikation auf einem Datensatz mit handschriftlichen Ziffern. So sind die Ziffern gut identifizierbar. Lediglich die „1“ ist stark abstrakt. Hier ist zu bemerken, dass dieses Bild als potentielle Eingabe aufgrund der Erzeugungsmethodik mit hoher Sicherheit in der Aktivierung eine 1 vorhersagt. Obwohl die 1 nicht zu erkennen ist, bedeutet das nicht unbedingt, dass eine Fehlfunktion vorliegt. Es muss erst geklärt werden, ob genau solch eine Ausprägung im Kontext der vorgesehenen praktischen Anwendung überhaupt auftreten kann.

Obwohl sich die vorliegende Arbeit mit Musikdaten befasst, wurden die theoretischen Grundlagen in diesem Abschnitt anhand von Fotos veranschaulicht, da die Theorie leichter mithilfe des intuitiven Wesens visueller Objekte (vgl. Abschnitt 1.3) verständlich gemacht werden kann. Die abschließende Analyse der KNNs für Musikdaten in Kapitel 7 soll vom Vergleich mit Erkenntnis-

sen aus der Fotodomäne profitieren. Visualisierungen von Daten, die ursprünglich nicht-bildlich sind, befinden sich per se schon auf einer interpretativen Ebene. So hat die interpretierende Analyse von Musikdaten im Gegensatz zu Fotos das gesteigerte Problem eine weitere Abstraktionsebene durchdringen zu müssen.

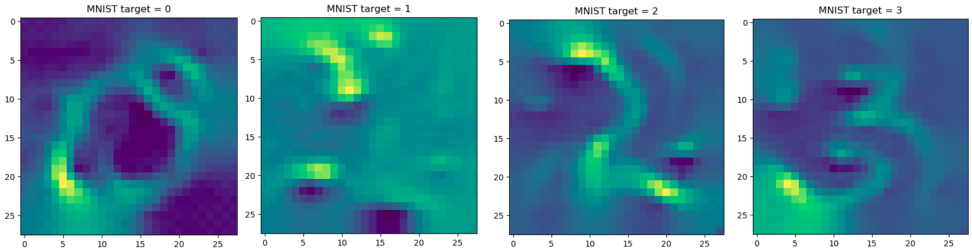


Abbildung 2.15.: Aktivierungs-Maximierung am Beispiel der Erkennung von handschriftlichen Ziffern aus dem Datensatz MNIST. [Grafik aus Ye, 2020]

3. Algorithmische Komposition

Der Begriff *Algorithmische Komposition* (AK, engl. *Algorithmic Composition*) beschreibt jede Form der Generierung von musikalischem Material unter Einbeziehung eindeutiger Handlungsvorschriften [vgl. Fernandez u. Vico, 2013, S.513]. Heute bezieht sich der Begriff meist auf die automatische Komposition durch Computerprogramme. Die ersten Anwendungen finden sich jedoch weit vor der Erfindung des Computers.

Im Jahr 1026 erfand Guido d'Arezzo (ca. 992–1050) die Methodik, Texte algorithmisiert durch feste Zuweisung von Silben zu Tonhöhen zu vertonen (als Variante seiner Technik der *Solmination* [Eggebrecht, 1967]). Ende des 18. Jahrhunderts erlangten die sogenannten *musikalischen Würfelspiele* als Zeitvertreib eine relativ hohe Popularität. Kurze musikalische Phrasen (meist eintaktig) werden durch Würfeln nach dem Zufallsprinzip ausgewählt und bilden aneinandergereiht ein einfaches Musikstück. Das bekannteste seiner Art ist W. A. Mozarts „Anleitung zum Componieren von Walzern vermittels zweier Würfel“ [KV, 1862, 294d/516f]. In den 1930er Jahren brachte Joseph Schillinger (1895–1943) dann mit seinem akademischen *Schillinger-System* [Schillinger, 1948] ein grundlegend mathematisches Kompositionssystem an amerikanische Musikhochschulen. Es beruht auf Parametrisierung von Kompositionen mit dem Ziel der Transformation. Die Erfindung der Computer im heutigen Sinne hat in dieser Tradition großes Interesse geweckt. So ist es für analytische Komponisten allzu reizvoll, mithilfe der Maschinen die zuvor erdachten Ansätze in weit höherer Geschwindigkeit auszuführen und auch bis dato undenkbar komplexe Ansätze zu realisieren. Eine Auflistung vieler weiterer bedeutender Meilensteine findet sich in Collins [2010, S.300].

Wichtig zu bemerken ist, dass die gesamte Grundlage für AK auf der Annahme beruht, dass sich Musik tatsächlich mathematisch Formalisieren lässt. So wird argumentiert, dass sich jedes Musikstück als Folge von Zahlenwerten definieren lässt, die komplexe Parameter wie Tonhöhe, Tondauer und Anschlagsstärke beschreiben. Solch ein Modell wird schnell hochkomplex, denn es unterliegt wie bereits in Abschnitt 2.1.2 erläutert der kombinatorischen Explosion.

Vorreiter und Wegbereiter einer stark formal mathematischen Abstraktion musikalischer Formen ist Xenakis [1971]. Es sei dazu gesagt, dass Xenakis durch seine intellektuelle Konsequenz, die ästhetische Standards und menschliche Erfassbarkeit schnell überwindet, in der Musikwelt ähnlich streitbar ist wie es zuvor zum Beispiel die Zwölftonmusik [Schönberg, 1992] war. Unbestritten ist jedoch der Einfluss, den sowohl Schönberg als eben auch Xenakis bis heute auf die kompositorische Praxis haben. So wurde eine analytische Herangehensweise an Komposition sowie ihre bedachte strukturelle Organisation wichtiger Bestandteil des künstlerischen Könnens. Gerade die Konsequenz macht Xenakis' Arbeit zu einem wertvollen Grundgerüst für die mathematische Modellierung in algorithmischen Kompositionssystemen.

Das Streben nach Abstraktion ist ebenso ein Leitmotiv des komplexen Problems des Algorithmisierens von Kompositionsprozessen. So nutzt Taube [2013, S.55ff] die Macht funktionaler

Programmiersprachen, um komplexe Prozesse zu organisieren. Der Komponist wird zum Steuermann kompositioneller Prozesse. Taube spricht später vom „Komponieren einer algorithmischen Komposition“ [vgl. Taube, 2013, p.65] und erreicht damit eine neue Freiheit. Auch der Ansatz der vorliegenden Arbeit basiert auf einer Metalösung des Problems. Nähere Erläuterung folgt in Kapitel 4.

Dieses Kapitel soll zunächst im folgenden Abschnitt 3.1 eine Kategorisierung verschiedener Ansätze und Methodiken darstellen. Aus der Vielfalt an Methodiken und bestehenden Ordnungsansätzen werden die zwei Grundkategorien der *regelbasierten* und *datenbasierten* Systeme als Kompromiss hergeleitet. Die in dieser Arbeit als Trainingsgrundlage ausgewählten Datensätze (Vorstellung folgt in Kapitel 5) sollen diese beiden identifizierten Kategorien abdecken. Zur näheren Erläuterung werden in den Abschnitten 3.2 und 3.3 populäre Methodiken dieser beiden Kategorien beispielhaft skizziert.

3.1. Kategorisierung

Es gibt einige Ansätze das vielfältige Gebiet der computergestützten Komposition zu ordnen. Ausgangspunkt einer Kategorisierung soll hier zunächst das Verhältnis Mensch zu Maschine bieten, wie es auch Fernandez u. Vico [2013, S.560] in ähnlicher Form beschreibt. Von Punkt zu Punkt emanzipiert sich das Programm vom Entwickler:

1. Handschriftliche Komposition
2. Komponist nutzt Notationsprogramm
3. Komponist programmiert stochastisches Regelsystem
4. Komponist modelliert heuristisches Wissen
5. Programm lernt Kompositionen nach vorgeschriebenen Prinzipien
6. Programm lernt aus Datenbank mit Kompositionen selbstständig

Punkt 2 ist heute der Standard für professionelle Komponisten. Die Eingabe, Veränderung und Vervielfältigung von Noten in WYSIWYG-Notationsprogrammen stellt im Vergleich zur Kopistenpraxis des Prä-Computerzeitalters (Punkt 1) einen riesigen Effizienzfaktor dar. Punkt 3 und 4 umschreiben eine Vielzahl an erfolgreichen Arbeiten im Umfeld von halbautomatischer Begleitung [Gannon, 1990], interaktiver Improvisation [Lewis, 2000] oder auch reaktiver Videospielmusik [Sweet, 2014]. Die Beschreibungen sind wage und austauschbar, da die konkrete Umsetzung solcher Programme einen großen kreativen Spielraum bietet. Arbeiten im Rahmen von Punkt 5 und 6 setzen fundierte Kenntnisse über Methoden des Maschinenlernens und der künstlichen Intelligenz voraus. Wichtig zu bemerken: Auch in diesen letzten Punkten erlangt die Maschine keine Unabhängigkeit vom Menschen. Mindestens die Datenbank muss dem Algorithmus vom Entwickler zugeführt werden.

Es gibt Definitionen von AK als einen formalen Prozess, um Musik mit nur minimalen manuellen Eingriffen zu erzeugen [vgl. Maurer, 1999]. In dem Kontext spricht man auch von *Automatischer Komposition* (engl. *Automated Composition*). Diese Definintion ist nach Ansicht des

Autors dieser Arbeit unter Kenntnis der historischen Arbeiten vom Anfang dieses Kapitels eher als subjektive Zielsetzung denn als methodische Beschreibung sinnvoll.

Von großem Interesse für diese Arbeit ist eine Einteilung nach methodisch-technischen Aspekten. Maurer [1999] identifiziert regelbasierte und datenbasierte Ansätze als grundlegend verschieden. Dies zieht eine lose Grenze zwischen Punkt 4 und 5. Zusätzlich führt Maurer [1999] die *stochastischen* Ansätze ein. Solche zufallsbasierten Kompositionen wurden von klassischen Komponisten wie Karl-Heinz Stockhausen (1928–2007) oder John Cage (1912–1992) vorangetrieben und später von Xenakis [1971] als *stochastische Musik* definiert. Der Autor dieser Arbeit identifizierte den Einsatz solch stochastischer Prinzipien sowohl innerhalb der regelbasierten Systeme (praktisches Beispiel folgt in Abschnitt 5.2.2) als auch klar in den datenbasierten Systemen wie in *Hidden Markov Models* (vgl. Abschnitt 3.3.1) sowie beim Lernverhalten von KNNs. Stochastik im Sinne von probabilistischer Entscheidungsfindung ist zwar oft das Mittel der Wahl in regelbasierten Systemen, jedoch ebenso grundlegendes Element in KI-System. Auf eine Abgrenzung von stochastischen Ansätzen wird aus diesem Grund im Folgenden verzichtet und stochastische Prozesse werden als allgegenwärtiges Element in AK gesehen.

Ames u. Domino [1992] unterscheiden zwei weitere interessante Kategorien, die die Intention des Entwicklers hinterfragen. So identifizieren sie das *empirical style modelling* als den Versuch, etablierte Kompositionsstrukturen und Klischees durch imitierende Algorithmen nachzubilden. Im Gegensatz dazu steht die *active style synthesis* als die durch künstlerisches Streben nach Neuschöpfung angetriebene Ergründung neuartiger Kompositionsstrukturen und -prinzipien. Der Algorithmus ist hier ein Impuls des Neudenkens.

Eine weitere besondere Gruppe, die nicht direkt in eine der beiden Kategorien regel- oder datenbasiert fällt, bilden die bio-inspirierten Algorithmen. Darunter fallen auch die KNNs, klassischerweise sind aber vor allem die *evolutionären Algorithmen* (EA) [Bäck u. a., 1997] gemeint, die unter der Metapher der biologischen Evolution [Darwin, 1859] Optimierungsprobleme approximieren. Darüber hinaus existieren weitere weniger populäre Biometaphern wie *Schwärme* oder *Ameisenkolonien* [vgl. Miranda u. Biles, 2007]. EA wurde bereits zahlreiche Male erfolgreich auf AK angewendet [Miranda u. Biles, 2007; Loughran u. O'Neill, 2020]. Der Clou ist dabei die Entkopplung des eigentlichen schöpferischen Prozesses der Musikgenerierung vom Prozess der Bewertung. In einem repetitiven Zyklus von Kombination von Kandidaten und anschließende Selektion durch eine Bewertungsfunktion nehmen die Resultate nach und nach gewünschte Formen an. Die Bewertungsfunktion kann dabei sowohl regelbasierte [Ostermann u. a., 2017] als auch datenbasierte Methodik (z.B. KNNs [vgl. Loughran u. O'Neill, 2020, Tab.1]) besitzen. EA ist also quasi auch eine Metamethode, die andere Ansätze kapseln kann. So ist das Vorhaben der vorliegenden Arbeiten dazu prädestiniert um später Anwendung in einem EA zu finden (mehr dazu in Kapitel 10).

3.2. Regelbasierte Systeme

Die Grundidee der klassischen regelbasierten Systeme ist es, vorgefertigte Heuristiken in einem Automaten zu implementieren. Es bietet sich an, bestehende Kompositionsregeln wiederzuverwenden, die bereits weit vor der Computerära in Kompositionslehren für Kompositionsstudenten formuliert wurden [z.B. Kühn, 1987]. In dieser Ausformung sind sie wissensbasierte Systeme im

Sinne des *empirical style modelling*. Es ist aber genauso möglich, neuartige Heuristiken für ein System zu entwerfen und so nie dagewesene Kompositionsformen im Sinne der *active style synthesis* zu entwickeln. Solch eine kreative Neuschöpfung wie sie sich Komponisten wie Xenakis [1971] oder auch Arvo Pärt [Hiller, 1996] zum Ziel gesetzt haben, punktet durch eine große Innovationskraft. Den regelbasierten Systemen sind zunächst keine Grenzen gesetzt. Die Kreativität des Programmierers bestimmt maßgeblich die Art, die Komplexität und die Qualität der Resultate. Das bedeutet vor allem die Existenz einer großen Bandbreite an Formen. Durch Kombination, Variation und Vermischung von Methoden erhöht sich die Vielfalt der Erzeugnisse weiter.

Abgegrenzt werden kann das Beschreibungsprinzip der *programmierten Partitur*. Dabei handelt es sich um ein Musikstück, dessen Ablauf bei der Aufführung der Partitur nicht festgelegt ist und deren konkrete Ausformungen variieren. Dieses Prinzip findet sich zum Beispiel auch in Stockhausens *Klavierstücke XI*, bei denen der aufführende Pianist die Form des Stückes zufällig abändert. Aufgrund der unüblichen Aufführungspraxis mit Computerprogrammen fällt hierbei auch öfter der Begriff der *Laptopmusik* [vgl. Collins, 2010, S.238].

Spontane Variationen sind vor allem Hauptaspekt der Jazzmusik. So ist Jazzimprovisation ein beliebtes Anwendungsfeld von AK. Oberkategorie bilden die *interaktiven Musiksysteme*. Interaktive Systeme sind meist weniger selbstständig und suchen ihre Stärken im entstehenden Dialog mit dem menschlichen Musiker. Durch clevere Reaktionsroutinen profitieren sie vom verspielten Geist des Mitmusikers [vgl. Ostermann u. a., 2017]. Das macht sie auch auf besondere Weise ansprechend und interessant für Zuhörer und Live-Publikum. So konnte sich beispielsweise das freie Improvisationssystem [Lewis, 2000], das tatsächlich auf relativ einfachen Heuristiken beruht, einer größeren Popularität erfreuen. Musikalische Interaktion ist deshalb in diversen öffentlichen Kunstinstallationen wiederzufinden [Collins, 2010, S.210f].

3.3. Datenbasierte Systeme

Datenbasiert bedeutet zunächst, dass einem System eine Datenbank mit Informationen zur Verfügung steht, aus denen es manuell, durch Heuristiken geleitet oder auch absolut selbstständig zum Beispiel Regeln, Grammatiken oder auch komplexe Verteilungen ableitet. Auch hier gilt, Kombinationen, Varianten und Vermischungen von Methoden tragen zur gewünschten Vielfalt der Erzeugnisse bei.

Grundsätzlich sind datenbasierte System meist eher dem *empirical style modeling* zuzuordnen. Sie kommen an ihre Grenzen, wenn Originalität verlangt ist. Ob dies eine Grenze zwischen datenbasierten und regelbasierten Formen zieht, ist nicht beantwortet und tangiert eher die (philosophische) Frage, ob Maschinen überhaupt kreativ sein können [Jacob, 1996, S.2f]. Prinzipiell gilt, dass ein Algorithmus Originalität durch Zufallsentscheidungen mindestens imitieren kann.

Ein berühmter Wegbereiter datenbasierter Musiksysteme ist David Copes *Experiments in Musical Intelligence* (*EMI*) [Cope, 1996]. Das System bekommt eine Datenbank von Kompositionen (meist eines einzelnen historischen Komponisten) in MIDI-ähnlicher Repräsentation. Es lernt dann die gemeinsamen Strukturen mithilfe von Methoden der Computerlinguistik und des *Pattern Matching*. Dann erzeugt es ein Stück, das eine Variation der ursprünglichen Kompositionen ist. Mathematisch gesehen befindet sich das neue Stück in einer Teilmenge des Definitionsraums, in dem sich eben die originalen Stücke befinden. So ist das Stück im musikalischen Sinne eher

eine Variation als ein Original [vgl. Hofstadter, 2001, S.293]. Der *künstliche Komponist*, hier *EMI*, kann keinen eigenen Stil entwickeln, wie es Cope [vgl. 2001, S.93] selbst beschreibt. Trotzdem führt das Vorgehen im Erfolgsfall dazu, dass die erzeugten Stücke mindestens als kohärent und im Bestfall in ihrer musikalischen Qualität nicht von den anderen Stücken der Datenbank unterscheidbar sind (mehr zu *EMIs* Erfolg folgt in Abschnitt 5.1.1).

EMI wird an vielen Stellen aktiv durch Cope oder durch von ihm implementierte Heuristiken geleitet. Im Folgenden sollen im Gegensatz dazu ein paar der populärsten vollautomatischen Generierungsmethodiken eingeführt werden. Darunter fallen die seit Mitte der 1970er beliebten *Hidden Markov Models* (HMM, Abschnitt 3.3.1) als auch die in den letzten Jahren dominierenden Varianten generativer KNNs (Abschnitte 3.3.2 und 3.3.3). Fernandez u. Vico [2013] bieten einen gründlichen kategorisierenden Überblick vieler weiterer bekannter Methodiken, die sich im Spannungsfeld zwischen AK und KI befinden.

3.3.1. Hidden Markov Models

Eine *Markov-Kette* ist ein stochastischer Prozess, der Wahrscheinlichkeiten zukünftiger Ereignisse durch Zustandsübergänge modelliert [Markov, 1913]. Die Übergangswahrscheinlichkeiten sind dabei konstant. So benötigt die Markov-Kette keinerlei Information über die Vergangenheit für eine Zukunftsprognose. Diese pragmatische, unter Umständen auch limitierende Eigenschaft wird *Markov-Eigenschaft* genannt.

In einem HMM ist die zugrundeliegende Markov-Kette verborgen [Rabiner u. Juang, 1986]. Die Ausgaben des Modells werden durch Zustandssignale beobachtet. Das Ziel ist durch Angabe einer Beispielsequenz auf eine mögliche Konstruktion der verdeckten Markov-Kette zurückzuschließen. Die verdeckten Parameter des Modells können auch mit dem erwartungsmaximierenden Baum-Welch-Algorithmus oder durch Gradientenverfahren gelernt werden [vgl. Rabiner u. Juang, 1986, S.11].

Klassische Anwendungsfelder in der Musikgenerierung sind vor allem Melodie- [Tipei, 1975] und Rhythmusgenerierung [Ames u. Domino, 1992] sowie interaktive Improvisation [Zicarelli, 1987]. Für eine Übersicht weiterer Arbeiten siehe Fernandez u. Vico [2013, S.536f]

3.3.2. Generative Neuronale Netze

Generative KNNs arbeiten nach den selben Grundprinzipien wie die Klassifikationsnetze aus Kapitel 2. Der wichtigste Unterschied liegt im Umgang mit den Trainingsbeispielen. So sind Musikdaten hier Ausgaben, die in verschiedensten Repräsentationen (vgl. Abschnitt 2.1) verdeckten Eingaben (unter Umständen zufällig bestimmt) zugeordnet werden. Wird das trainierte Netz später mit einer beliebigen unbekannten Eingabe ausgeführt, so sind die Ausgaben Interpolationen verschiedener Ausprägungen der Beispieldaten [vgl. Fernandez u. Vico, 2013, S.541].

Ebenso flexibel wie die Netze sind aber auch die Anwendungsvarianten. Aber auch diverse andere Konzepte sind denkbar, zum Beispiel schlägt Lewis [1991] die rekursive Verfeinerung ausgehend von einer Zufallskomposition vor. Klassische Anwendungsfelder sind Melodiegenerierung [Todd, 1988], Harmonisierung [Shibata, 1991] und Jazz-Improvisation [Nishijima u. Watanabe]. Für eine Übersicht weiterer Arbeiten siehe Fernandez u. Vico [2013, S.541f].

3.3.3. Generative Adversarial Network

Die *Generative Adversarial Networks* (GAN) [Goodfellow u. a., 2014] sind ein Sonderfall der generativen KNNs. So besteht ein GAN immer aus zwei KNNs, die ein *Nullsummenspiel* durchführen, wie es aus der Spieltheorie bekannt ist. Das eine KNN ist der *Generator*, der aus einer Zufallseingabe Ausgaben erzeugt. Das zweite KNN, der *Diskriminator*, bewertet die Ausgaben des Generators, indem er lernt diese von einer Menge an gewünschten Beispielen zu unterscheiden. Die bewerteten Ausgaben sind dann wiederum Trainingseingabe für den Generator. Ziel der wechselseitigen Optimierung ist es, dass der Diskriminator die Ergebnisse des Generators nicht mehr von den echten Beispielen unterscheiden kann.

Ergebnisse sind in vielen Anwendungsfeldern überaus erfolgreich. So verblüffen mit diesem Ansatz generierte Portraits [Karras u. a., 2018] mittlerweile durch ihren Realismus (Stichwort: *DeepFake*). Aktuelle Arbeiten zur Generierung von Musik mit GANs wie *WaveGAN* [Donahue u. a., 2019] oder *MuseGAN* [Dong u. a., 2018] zeigen das Potential, dass sich in naher Zukunft zunächst einmal hyperrealistische Klänge produzieren lassen. Wie weit der Weg zur hyperrealistischen Komposition ist, wird die Zeit zeigen.

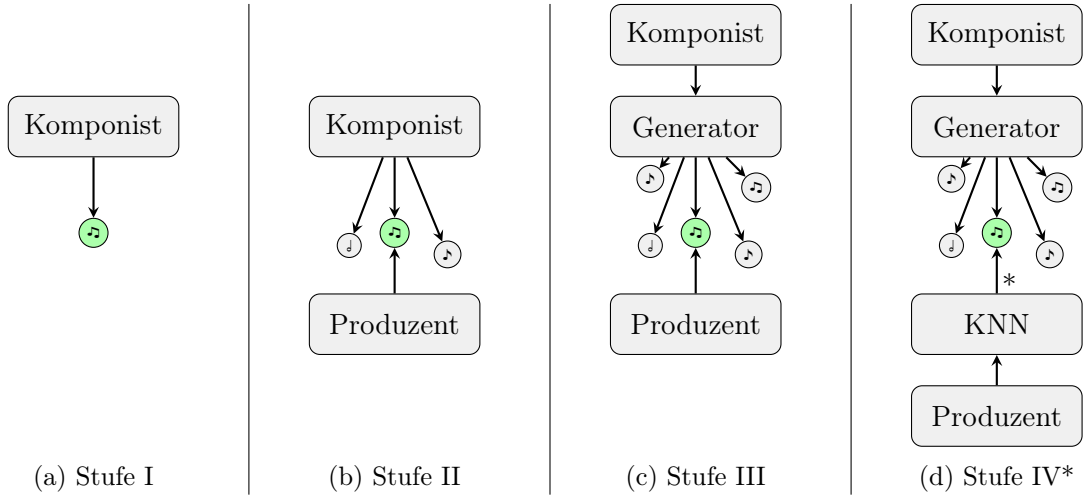
Teil II.

Praktische Umsetzungen

4. Methapher des künstlichen Produzenten

„Es gibt zwei Arten von Musik, gute und schlechte. Ich spiele die gute.“
Louis Armstrong (1901–1971)

Wie in der Einleitung eingeführt, sollen bestehende Arbeiten *künstlicher Komponisten*, wie sie im vorherigen Kapitel beschrieben wurden, durch das Sortieren ihrer Erzeugnisse in die Klassen $\smile$ („gefällt“) und $\frown$ („gefällt nicht“) verbessert werden. Der sortierende Algorithmus ist ein KNN, das hier und im Folgenden als *künstlicher Musikproduzent* bezeichnet wird. Das Konzept ist nach Ansicht des Autors der vorliegenden Arbeit der logische nächste Schritt in einer beobachtbaren Entwicklung, die in drei etablierte Stufen I–III und die neue Stufe IV* eingeteilt werden kann (siehe Abbildung 4.1). Da dabei die Leistung einer bestehenden Implementierung ohne die Modifikation des eigentlichen Algorithmus optimiert wird, handelt es sich hierbei um ein Metaoptimierungsproblem.



(a) Stufe I (b) Stufe II (c) Stufe III (d) Stufe IV*

Abbildung 4.1.: Weiterentwicklung der Zusammenarbeit von Komponisten und Musikproduzenten in vier Stufen. Der Generator ist ein *künstlicher Komponist*, das KNN der *künstliche Musikproduzent*.

In der ersten Stufe ist der Komponist ein selbstständiger menschlicher Arbeiter. Er schreibt an einem Stück und veröffentlicht es. In Stufe II schaltet sich ein Produzent ein. Der Komponist bietet eine Vielzahl an Stücken an und der Produzent wählt eines nach eigenem Geschmack aus.[1] Stufe III ist der aktuelle Stand der Entwicklung. Der *künstliche Komponist* (Generator) kann jedwede Art von Computerunterstützung beim Musikproduktionsprozess sein, beispielsweise eine *Digital Audio Workstation* (DAW). Diese hat Funktionen wie automatisches Kopieren, automatische Tonhöhenkorrektur oder intelligente Vorlagen. Der Komponist kann so mehr Stücke

[1] In der realen Welt spielen meist auch kommerzielle Faktoren eine Rolle. Das kann als Vorhersage eines kollektiven Musikgeschmacks interpretiert werden.

in kürzerer Zeit erzeugen. Die größere Auswahl führt auf der anderen Seite zu einem erheblichen Mehraufwand beim Produzenten. Im Bereich der Forschung kann der *künstliche Komponist* in Stufe III von einem halbautomatischen (d.h. der Komponist steuert) bis zu einem vollautomatischen System (d.h. der Komponist ist der Programmierer) reichen. Der Produzent ist in der wissenschaftlichen Welt ein Forscher, der Beispielstücke zur Präsentation eines entwickelten Musiksystems in Vorträgen und Artikeln auswählt.

Da ein (voll-)automatisches System mühelos eine enorme Anzahl an Stücken erzeugen kann, wächst die Arbeit des Produzenten von Stufe II auf Stufe III ebenso enorm. Der *künstliche Produzent* schafft hier Entlastung, indem er die Stücke mindestens vorsortiert. Er ordnet sich in Stufe IV* zwischen dem *künstlichen Komponisten* und dem Produzenten ein und ist Letzterem ein Hilfswerkzeug. Indem schlechte Stücke gar nicht erst gesichtet werden müssen, verringert sich der Zeitaufwand. Das KNN arbeitet wie ein Sieb für gute Stücke. In der Konsequenz verbessern sich die Erzeugnisse des *künstlichen Komponisten*.

Dieser Vorschlag bedeutet nicht, dass alle Musik künftig von Computern bewertet werden kann, muss oder soll. Ebenso wie Genres sich nicht ablösen, sondern neue Genres sich zu alten Hörgewohnheiten hinzugesellen, und so wie es bisher auch weiterhin selbstständige Komponisten ohne Produzenten gibt (Stufe I), werden alle Abstufungen des Konzepts I–IV* in Zukunft stets sinnvoll Anwendungen finden. Stufe IV* öffnet lediglich eine neue Tür.

4.1. Abgrenzung zu Generative Adversarial Networks

Die Schematik von Stufe IV* aus Abbildung 4.1d besitzt eine gewisse ähnlich zu dem Prinzip des GAN aus Abschnitt 3.3.3. So gibt es hier mit dem *künstlichen Komponisten* ebenso einen Generator, dessen Ausgaben durch einen Diskriminator, nämlich den *künstlichen Produzenten*, bewertet werden. Allerdings findet keine Rückkopplung statt. Im Gegensatz zum GAN wird die Funktionsweise des Generators nicht durch die Aussagen des Diskriminators beeinflusst. Sie können es auch gar nicht, denn bestehende Arbeiten, die nicht auf maschinellem Lernen basieren, sind nicht automatisch optimierbar. So zielt der vorliegende Ansatz nicht darauf ab, die Funktionsweisen bestehender Generatoren, sondern die möglichen Resultate durch Selektion zu verbessern. Das ist auch insofern von Vorteil, da regelbasierte und bereits vorevaluierte Arbeiten oftmals schon vor der Optimierung eine höhere Qualität haben als pure datenbasierte Ansätze. So kann auch der schöpferisch originelle Impuls des menschlichen Komponisten, der ein AK-Programm entwickelt hat, erhalten werden (vgl. Kapitel 3). Zudem können durch die spätere Analyse der Entscheidungsprinzipien, die die KNNs beim Training ausprägen, Rückschlüsse auf die verbesserungswürdigen Aspekte der Generatoren gezogen werden (vgl. Kapitel 7).

4.2. Abgrenzung zu Empfehlungssystemen

Der Ansatz des *künstlichen Produzenten* hat einen klaren Bezug zum Forschungsgebiet der *Empfehlungssysteme*. So ist das System nach Definition [vgl. Jannach u. a., 2010] insofern ein Empfehlungssystem, dass es versucht, eine individuelle Vorhersage über das mögliche Feedback eines Nutzers zu einem konsumierbaren Objekt zu treffen. So findet sich auch das Ziel der Informationsreduktion aus einer unüberschaubaren Menge an Möglichkeiten bei beiden (vgl. Abbildung 4.1d).

Die maschinelle Vorhersage von individuell angepassten Listen mit Musikstücken (Playlists) ist ein populäres Forschungsfeld innerhalb der Musikempfehlungssysteme. Die Variablen, die das Gefallen eines Stückes auf Objektebene beeinflussen, sind dabei vielfältig. Historisch gesehen spielen die Prinzipien der Dissonanz (aufbauend auf Helmholtz [1865] und Stumpf [1883]) eine wichtige Rolle. Noch heute wird „dissonant“ als Synonym für „unschön“ benutzt („das hört sich schräg an“). In der multivalenten Musikkultur einer globalisierten Welt verlieren solche Zusammenhänge jedoch endgültig an Relevanz.

Der entscheidende Unterschied zur vorliegenden Arbeit besteht in der modernen Praxis erfolgreicher Empfehlungssysteme. Meist sind dies kommerzielle Anwendungen, die in ihrer Intention vom *künstlichen Produzenten* klar verschieden sind. Als wissensbasierte Systeme nutzen sie typischerweise den Vergleich vieler Kaufentscheidungssequenzen verschiedenster Nutzer, um daraus die konkrete individuelle Entscheidung herzuleiten [vgl. Jannach u. a., 2010, S.4ff]. Zudem bedienen sich aktuell erfolgreiche Systeme an Erkenntnissen aus Psychologie und Soziologie. So treten Metainformation wie soziales Umfeld [Gebesmair, 2001], kultureller Hintergrund oder interkulturelle Verbindungen [Patel u. Demorest, 2013] in der Vordergrund der Vorhersage. Bei Popularmusik, die in besonderem Maße ein soziales Element darstellt, haben sich diese Merkmale als enorm aussagekräftig herausgestellt [Laplante, 2014].

An einem einfachen Beispiel wird klar, wie der Mensch seinen individuellen Geschmack als soziales Element benutzt: Es entsteht ein Gemeinschaftsgefühl zum Beispiel unter den Besuchern eines Konzertes oder den Fans einer Band (prominenterweise kommuniziert durch ein passendes T-Shirt). So ließe sich auch eine Klassifizierung durchführen: Eine Geschmacksklasse ist die Menge aller Individuen, denen bestimmte Musikstücke, eine bestimmte Band oder ein bestimmter Komponist gefallen oder eben auch nicht gefallen. Diese Faktoren finden aber beim *künstlichen Produzenten* als musikdatenbasierte Konstruktion keine Anwendung.

4.3. Abgrenzung zur Hit-Prediction

Eine Sonderform der Empfehlung ist die Problemdomäne der *Hit Song Prediction*. Dabei wird versucht, meist aus der Menge an vergangenen Top10- oder Top100-Charts, das Hit-Potential unbekannter Songs algorithmisch zu bestimmen. Als Datengrundlage bieten sich musikalische Charakteristiken wie Tonart, Tempo oder Länge und höherwertige Merkmale wie Energie, Tanzbarkeit oder Stimmung sowie auch Metainformationen wie Datum der Veröffentlichung, Songtexte oder Plattencover an [Middlebrook u. Sheik, 2019]. Hier besteht vor allem finanzielles Interesse durch die Musikindustrie, die jährlich sehr viel Geld in die Suche neuer Talente investiert [vgl. Herremans u. a., 2014]. Der größte Unterschied zum Ansatz der vorliegenden Arbeit besteht allerdings in der ausschließlichen Betrachtung menschengemachter Popularmusik. Die vorliegende Arbeit sieht ihre Stärke gerade in der Auseinandersetzung mit AK.

4.4. Abgrenzung zur automatischen Genre-Klassifikation

Genre ist ein populäres Klassifikationsproblem. Es bietet die Möglichkeit verschiedenste Methoden der MIR-Forschung wie Tempo- und Tonhöhenerkennung oder auch höherwertige Merkmale

wie Instrumentierung oder gar implizierte Emotionen sinnvoll zu kombinieren. Die Einteilungsklassen sind dabei durch eine Vielzahl an musikwissenschaftlichen Arbeiten begründet. Diese scheinbare Objektivität ist aber auch oft die Fallhürde der Methoden. So steht und fällt die Güte der Klassifikation stets mit der Korrektheit der Annotationen in den Trainingsdaten. Bei Genres ist die objektive Begründbarkeit aber oft strittig.

Moderne Ansätze zeigen, dass Metainformation über kulturelle und soziale Zusammenhänge sowie persönliche und demografische Merkmale auch hier erfolgreich genutzt werden können [Laplante, 2014]. Die von Musikwissenschaftlern auf Grundlage von qualitativ definierten Merkmalen vorgenommene Einteilung von Musikstücken in Genres wird auch von Hörern gern genutzt. Sie leiten Aussagen ab wie „Ich mag klassische Musik“, „Mir gefällt Jazz-Funk“ oder auch „Ich hasse Popmusik“. Obwohl Genre wissenschaftlich als auch im Alltag ein gern genutzter Klassifizierungsansatz ist, sind Genres aber oft unzulänglich, vor allem bei der Vorhersage des individuellen Geschmacks. Niemals gefallen einem Individuum uneingeschränkt alle Stücke eines Genres. Und objektiv festgelegte Genres müssen nicht dem individuellen (gefühlten) Verständnis entsprechen.

Unbestritten hingegen ist, dass die durch Genres implizierte Ähnlichkeitsbeziehung zwischen musikalischen Objekten existiert: „Wer Haydns Streichquartette mit Genuss hört, wird mit großer Wahrscheinlichkeit auch jene von Mozart mögen.“ [Gebesmair, 2001, S.13] So kann die vorliegende Arbeit durchaus von der Forschung zur automatisierten Klassifikation von Genres lernen. Die in Abschnitt 6.2 gewählten Architekturen bedienen sich konsequenterweise etablierter KNN-Architekturen aus unter anderem diesem Forschungsgebiet.

5. Datensätze

Ein *künstlicher Produzent* benötigt immer einen *künstlichen Komponisten*, dessen Werke er lernen und bewerten kann. Zur Machbarkeitsstudie der Idee aus Kapitel 4 führt diese Arbeit Experimente für insgesamt zwei Generatoren durch. Dabei wurde die Auswahl auf Grundlage der Verfügbarkeit und Portabilität der Systeme beziehungsweise deren Quellcodes, der Aktualität der Arbeiten sowie der in Kapitel 3 eingeführten Unterteilung in datenbasierte und regelbasierte Generatoren getroffen. Für die Evaluierung wurde der datenbasierte *DeepBach* und der regelbasierte *Computoser* gewählt. Dieses Kapitel erklärt ihre jeweilige Funktionsweise und erläutert die konkrete Erstellung der zum Training und Testen genutzten Datensätze. Zudem erfolgt eine Einordnung in den Kontext verwandter Arbeiten mit näherer Begründung der Wahl.

5.1. DeepBach

DeepBach [Hadjeres u. a., 2017] ist ein *künstlicher Komponist*, der vierstimmige Choräle im Stil von J. S. Bach (1685–1750) imitiert. Zu Lebzeiten hat Bach 389 Choräle komponiert [Bach, 1985], die allesamt im Vokalsatz SATB (Sopran-Alt-Tenor-Bass) stehen, einen $\frac{4}{4}$-Takt haben, ähnliche barocke Harmonieverläufe aufweisen und vergleichsweise kurz sind. Diese große Sammlung an ähnlichen Stücken überschaubarer Komplexität rückt für datenbasierte Ansätze regelmäßig in den Blickpunkt des Interesses. Die Sammlung konnte für diese Arbeit über das Python-Toolkit *music21* [Cuthbert u. Ariza, 2010] heruntergeladen werden. Die Sammlung wurde dabei um die Stücke reduziert, die von den zuvor beschriebenen Eigenschaften abweichen. Die übrigen 369 Stücke werden im Folgenden als die BACH_{369} bezeichnet.

Der folgende Abschnitt 5.1.1 gibt zunächst einen Überblick über verwandte Ansätze. Die meisten davon widmen sich ebenso der Choralimitation. Die Funktionsweise vom gewählten System *DeepBach*, das selbst ein KNN-Ensemble ist, wird in Abschnitt 5.1.2 erklärt. Die Konstruktion einer Trainings- und Testmenge für diese Arbeit stellt Abschnitt 5.1.3 dar.

5.1.1. Verwandte Arbeiten

Stücke im Stile bekannter Komponisten zu komponieren ist eine typische Lernaufgabe auch für menschliche Kompositionsschüler. Diese Aufgabe von einem Computer übernehmen zu lassen ist demnach ein naheliegender Gedanke (vgl. Abschnitt 3.1: *empirical style modelling*). BACH_{369} besticht hier gerade durch die Konkretheit und Beschränktheit der Daten. Ähnliche Arbeiten, die sich dem Imitieren kompletter Werkesammlungen von Bach oder auch anderen Großmeistern wie Mozart, Chopin oder Bartók widmen, liefern aufgrund der höheren Varianz in der Menge der Stücke meist schlechtere Imitationen.

Die erste Arbeit zur vollautomatischen Erzeugung von Bachchorälen ist Ebcioğlu [1988]. Das zugrundeliegende regelbasierte Expertensystem bekommt eine Melodie vorgegeben und versucht

eine sinnvolle Bach-ähnliche Harmonisierung zu erzeugen. Dazu wurden etwa 350 heuristische Kompositionsvorschriften implementiert, die durch Fehleranalyse und -behebung nach und nach die Erstellung eines *korrekten* Chorals gewährleisten sollen. Trotz des großen Aufwands ein adäquates Regelwerk zusammenzustellen, sind die Erfolge aus heutiger Sicht überschaubar. Zudem ist das Programm, dessen letzte Version zwar als logische Programmierung in PROLOG implementiert war [Ebcioğlu, 1990], auf aktuellen Systemen ohne enormen Portierungsaufwand nicht mehr lauffähig.

Den ersten Versuch mithilfe von KNNs das Problem zu lösen, stellt das Projekt *HARMONET* [Hild u. a., 1992] dar. Eine Pipeline von mehreren KNNs mit unterschiedlichen Aufgaben analysiert zunächst harmonische Strukturen der Melodie, entwirft anschließend passende Akkorde und fügt abschließend Verzierungen durch weitere Achtelnoten hinzu. Für die damalige Zeit waren die Ergebnisse zufriedenstellend. Aufgrund der Begrenzung der Trainingsmenge auf ein paar Dutzend Choräle und einem betrachteten Fensterausschnitt von nur einem Takt, was beides auf die damaligen Beschränkungen der Rechenleistung zurückzuführen ist, wurde *HARMONET* von späteren Arbeiten rasch überholt.

Ein ähnlicher Ansatz auf Basis eines HMM ist Allan u. Williams [2004]. Die sichtbaren Zustände repräsentieren Melodienoten, die verdeckten Zustände Akkorde. Die Arbeit ist inspiriert von *HARMONET*, wurde aber ohne jegliche heuristischen Vorbedingungen konzipiert. Zum Training werden weniger als 200 Stücke genutzt, da das Modell entweder Stücke in Dur oder in Moll erzeugen soll. Die Ergebnisse sind entsprechend nicht optimal. Beispiele sind zudem nicht mehr zugänglich.

Eine hingegen äußerst erfolgreiche und beeindruckende Arbeit ist der *Bachbot* [Liang u. a., 2017]. Beispiele dieser Arbeit sind bestechend überzeugend und bestehen den musikalischen Turing-Test mit einer Abweichung vom puren Raten von 1 %. Das Modell nutzt eine gestapelte LSTM-Architektur und verteilt Trainings- und Testmengen auf allen verfügbaren Bachchorälen, wobei diese im Vorfeld einheitlich in die Tonart C-Dur (beziehungsweise deren Parallele a-Moll) transponiert werden. Das Problem für die Einbeziehung von *Bachbot*-Chorälen in die vorliegende Arbeit besteht darin, dass das konkrete Programm, also die gelernten Gewichte des KNNs, nicht veröffentlicht sind. Online[1] findet sich der Quellcode samt Anleitung zum Trainieren eines eigenen *Bachbots*. Aufgrund des großen Aufwands dieser Möglichkeit und der Unvergleichbarkeit mit dem konkret evaluierten KNN aus Liang u. a. [2017] wurde im Rahmen dieser Arbeit davon abgesehen.

Dasselbe gilt für die Anwendung von *WaveGAN* [Donahue u. a., 2019]. Dieses wurde mit gewissem Erfolg auf Bach Pianoaufnahmen trainiert, bietet zudem, aufgrund der höheren Komplexität des Ansatzes sowie der nicht ausschließlichen Ausoptimierung auf Bach, weit schlechtere Ergebnisse als *Bachbot* und *DeepBach*. Zudem werden bei diesem Ansatz direkt Audiodaten erzeugt, das heißt es steht hier keine symbolische Repräsentation zur Verfügung.

Eine sehr populäre und vieldiskutierte Arbeit einer Generalisierung der Stilimitation ist das bereits in Abschnitt 3.3 beschriebene System *EMI* [Cope, 1996]. Es wird mit Stücken eines Komponisten gefüttert, dessen Stil es auf Grundlage statistischer Analysen lernt und mögliche Variationen reproduziert (vgl. Abschnitt 3.3). In Aufführungen wurde die Qualität durch

[1] `https://github.com/feynmanliang/bachbot` (zul. abgerufen am 9.6.2021)

Musikliebhaber und Musikwissenschaftler in einem musikalischen Turing-Test [Ariza, 2009] als bestechend authentisch aufgenommen. Dabei muss allerdings bemerkt werden, dass Cope bei den Aufführungen und Veröffentlichungen der Stücke[2] stets selbst eine Auswahl der besten Stücke getroffen hat. Damit hat er ebenjene Aufgabe, die in dieser Arbeit als die des Produzenten bezeichnet wird (vgl. Abbildung 4.1c), selbst übernommen.

Ein Projekt zur Analyse und Verifikation von *EMI* war die Sammlung *5000 Works in Bach Style* [Cope, 2012], die unveränderte Ausgaben von Copes Programm verfügbar machte. Diese wäre für die vorliegende Arbeit von großem Interesse gewesen. Die Stücke sind jedoch nicht mehr länger abrufbar und nach direkter Rückfrage bei Cope derzeit leider nicht auffindbar und auch nicht problemlos reproduzierbar.

5.1.2. Funktionsweise

DeepBach [Hadjeres u. a., 2017] besteht aus insgesamt vier Generatoren p_1, p_2, p_3, p_4 für die vier Stimmen des SATB, wobei jeder Generator selbst ein Ensemble von vier einzelnen KNNs ist. Die originalen Choräle werden in eine eigensentworfene sequentielle Textrepräsentation gebracht (vgl. Abbildung 5.1, oben). Als Teilproblem wird die Vorhersage einer einzelnen Viertelnote einer einzelnen Stimme $i \in \{S, A, T, B\}$ definiert. Bekannt sind dabei die Noten der anderen Stimmen zum selben Zeitpunkt sowie ein begrenzter Ausschnitt der vorherigen und der nachfolgenden Noten jeder der vier anderen Stimmen.

Diese vergangenen beziehungsweise zukünftigen Ausschnitte werden von zwei unabhängigen rekurrenten Netzen verarbeitet. Diese bestehen jeweils aus zwei seriell-gestapelten LSTM-Einheiten (identisch mit Tabelle 6.3c aus dem folgenden Kapitel) und werden von den Autoren mit *Deep RNN* bezeichnet (vgl. Abbildung 5.1).

Die Noten der anderen Stimmen der aktuellen Viertelnote werden hingegen aufgrund der fehlenden sequentiellen Information von einem einfachen MLP (vgl. Abbildung 5.1: *Neural Network*) verarbeitet. Die letzte Ausgabe der zwei *Deep RNNs* und die Ausgabe des MLP werden an ein weiteres zusammenfassendes MLP gegeben, welches dann die eigentliche Vorhersage der gewünschten Note trifft. Vorteil ist hier, dass der beschränkte Tonumfang der jeweiligen Vokalstimme die Ausgabe auf ungefähr zwei Oktaven (≈ 24 Töne) beschränkt.

Zur Generierung eines Stücks wird der *pseudo-Gibbs sampling*-Algorithmus [Geman u. Geman, 1984] genutzt. Leicht vereinfacht gesagt, bekommt dieser als Eingabe eine Länge L (Anzahl an Viertelnoten) und eine maximale Anzahl an Iterationen M. Zunächst werden vier Listen $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$, $\mathcal{V}_4$ der Länge L für die vier Stimmen SATB angelegt. Jeder Eintrag wird zufällig-gleichverteilt aus den für die jeweilige Vokalstimmen möglichen Tonhöhen initialisiert. Dann beginnt in M-facher Iteration die schrittweise Optimierung dieser bis dahin „chaotischen Komposition“. In jedem Schritt wird aus einer zufällig-gewählten Liste $\mathcal{V}_i$ mithilfe des passenden KNN-Ensembles p_i ein zufällig-gewählter Eintrag verändert. So entsteht nach und nach aus dem Chaos eine „sinnvolle Komposition“. Für einen erfolgreichen Durchlauf muss M entsprechend größer als L gewählt werden. Alle Details des Algorithmus finden sich in Hadjeres u. a. [2017, Alg.1].

Mit jedem Durchlauf des Algorithmus gibt der *künstliche Komponist* also einen neuen Choral im Stile von J. S. Bach aus. Der Code von DeepBach mitsamt der Modellparameter und -gewichte

[2] z.B. David Cope (2003): *Virtual Bach – Experiments in Musical Intelligence* (Centaur, Record Nr.: CRC 2619)

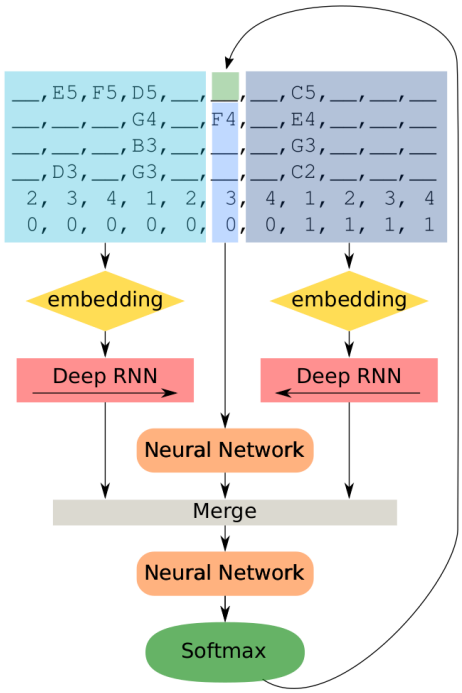


Abbildung 5.1.: Grafische Darstellung des KNN-Ensembles aus *DeepBach* zur Vorhersage der aktuellen Note der Sopranstimme. [Grafik aus Hadjeres u. a., 2017, Abb.4]

ist online veröffentlicht[3]. Standardwerte sind darin $L = 64$ und $M = 500$. Damit wird jeder Eintrag (d.h. jede Note) im Mittel $\frac{M}{4L} \approx 1{,}95$-mal optimiert. Das Modul besitzt darüberhinaus eine praktische Integrationsmöglichkeit in das offene WYSIWYG-Notensatzprogramm *MuseScore*[4]. Darin agiert es als interaktive Kompositionshilfe, indem es dem oben beschriebenen Algorithmus unveränderliche Einträge vorschreibt.

5.1.3. Sammlung gefälschter Choräle

Um einen Trainingsdatensatz für das maschinelle Lernen zu erhalten, wurden mithilfe von *DeepBach* neue Stücke generiert. Als direkte Ausgangsrepräsentation wurden zunächst MIDI-Dateien erzeugt. Um eine Vergleichbarkeit nicht nur mit einzelnen Werken, sondern mit der gesamten Choralsammlung BACH_{369} zu gewährleisten, wurden als Trainingsmenge 369 neue Stücke auf einmal generiert. Dabei wird für jedes Stück in BACH_{369} ein Gegenstück derselben Länge L erzeugt. Die Anzahl der Iterationen M wird für jedes neue Stücke proportional zu seiner Länge L als $M = q \cdot 1{,}95 \cdot 4L$ gewählt, um die einzelnen Stücke bei der Generierung gleichwertig zu optimieren. Dabei wird die Qualität der Stücke mit $q = 3$-mal so hoch gewählt wie standardmäßig vorgeschlagen, da besonders gute Stücke gesucht sind. Diese neue Sammlung an Computer-generierten Chorälen mit einer identischen Längenverteilung wie bei BACH_{369} soll im Kontext der folgenden Experimente als eine Fälschung interpretiert werden. In der Konsequenz wird sie als qualitativ schlechter angesehen. Die gefälschte Sammlung wird hier und im Folgenden analog zum Begriff BACH_{369} als FAKE_{369} bezeichnet.

[3] `https://github.com/Ghadjeres/DeepBach` (zul. abgerufen am 9.6.2021)
[4] siehe `musescore.org` (zul. abgerufen am 9.6.2021) und `github.com/musescore/MuseScore` (zul. abgerufen am 9.6.2021)

Es sei darauf hingewiesen, dass die Choräle von Bach keine Tempoangabe besitzen. Alle Stücke (echte wie Fälschungen) werden in den Tempo-abhängigen Repräsentationen mit 100 BPM abgespielt. Zur Audio-Generierung wurde der online[5] frei verfügbare Software-Sampler *The Leeds Town Hall Organ* verwendet. Exakt dieser wurde auch in Hadjeres u. a. [2017] für Hörstudien genutzt, sodass eine spätere Vergleichbarkeit gewährleistet ist. FAKE$_{369}$ und auch BACH$_{369}$ finden sich im MIDI-Format in der digitalen Anlage dieser Arbeit. Darüberhinaus sind beispielhaft ein paar Stücke in der angesprochenen Orgelumsetzung beigefügt.

5.2. Computoser

Der *Computoser* [Bozhanov, 2014] ist ein *künstlicher Komponist*, der mithilfe eines heuristischen Regelsystems Musikstücke erzeugt. Der Anspruch des Projekts war dabei von Beginn an nicht die Erzeugung von Top10-Hits oder künstlerisch wertvollen Resultaten. Stattdessen wurde ein großer Aufwand in die Präsentation der Stücke gesteckt. Die Website `Computoser.com` präsentiert dem Besucher eine Vielzahl an *Computoser*-Stücken in einem Webplayer (siehe Abbildung 5.2). Dieser Player ermöglicht Vorspulen und Überspringen von Stücken und wählt dabei zufällig eins aus 8007 verfügbaren Stücken aus. Der Nutzer hat zudem die Möglichkeit ein positives oder negatives Geschmacksurteil über das angehörte Stück mithilfe eines *Like*- beziehungsweise *Dislike*-Buttons abzugeben. Die Wertungen werden im Server akkumuliert und im Falle einer positiven Summe dem Besucher als allgemeines Urteil angezeigt.

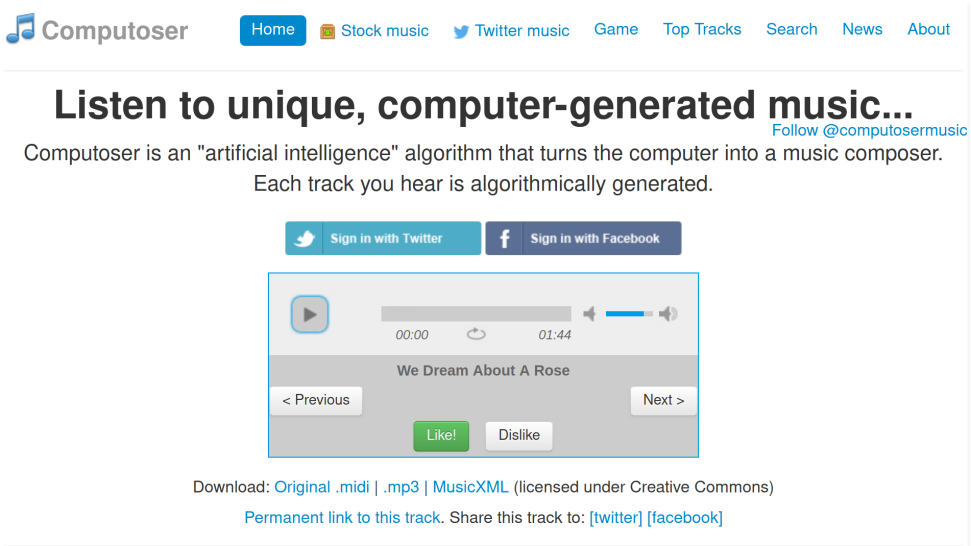


Abbildung 5.2.: Oberfläche der Website `Computoser.com`.

Der Autor vom *Computoser* hat die Datenbank der gesammelten *Like*- und *Dislike*-Einzelurteile dem Autor dieser Arbeit zugänglich gemacht. Nachdem verwandte Arbeiten und die Funktionsweise des *Computoser*s in den folgenden beiden Abschnitten vorgestellt und erklärt sind, diskutiert Abschnitt 5.2.3 wie aus den Daten ein binäres Geschmacksurteil abgeleitet werden kann.

[5] `https://www.samplephonics.com/products/free/sampler-instruments/the-leeds-town-hall-organ` (zul. abgerufen am 9.6.2021)

5.2.1. Verwandte Arbeiten

Der *Computoser* hat keinen so klar absteckbaren Anwendungsbereich wie das zuvor beschriebene *DeepBach* mit seiner Beschränkung auf Bachchoräle. Jeder regelbasierte Musikgenerierungsansatz (siehe Abschnitt 3.2) weist potentiell bestimmte Ähnlichkeiten zum *Computoser* auf. Als besondere Zielsetzungen betont Bozhanov [2014] allerdings, mit dem *Computoser* einen höheren Variantenreichtum, stärkeres Memorieren der Stücke sowie eine höhere allgemeine Gefälligkeit als bisherige Ansätze erreichen zu wollen.

Bozhanov [2014, Abs.7] stellt auch direkt eine Vergleichsstudie vom *Computoser* mit anderen Ansätzen vor, in denen Hörer nach der Vorführung von Beispielstücken schlicht die Frage nach dem Gefallen beantworten sollten. Dabei sagten 42 % der Probanden, dass sie *Computoser* mögen würden. Der evolutionäre *DarwinTunes* [MacCallum u. a., 2012], dessen Evaluierungsprozess von knapp 7000 Projektteilnehmern per Votingverfahren gesteuert wurde, gefiel ebenso 42 %. Die anderen Arbeiten überzeugten weniger: *SoundHelix* [Schürger, 2016], quasi eine programmierte Partitur (vgl. Abschnitt 3.2) für *Electronic Dance Music*, erreichte 34 %, der zeitgenössisch-klassische *künstliche Komponist Iamus* [Diaz-Jerez, 2011] lag bei 18 %, das Zellularautomat-basierte *Wolfram Tones* [Wolfram, 2002] bekam 11 % und das mathematisch-abstrakte *Fractal Music* [Hazard u. Kimport, 1999] lediglich 7 %.

Als nicht regelbasiertes System sei als verwandte Arbeit *MuseGAN* [Dong u. a., 2018] zu nennen. Dieses System basiert auf einem Convolutional GAN, das auf einer Trainingsmenge von knapp 175 000 echten Musikstücken in Pianoroll-Repräsentation aus dem *Lakh MIDI Dataset* [Raffel, 2016] das Erzeugen von Popmusik-ähnlichen Musikstücken zu erlernen versucht. Einschränkung ist auch hier vor allem, dass keinerlei Bewertungen vorhanden sind, sowie dass trotz Verfügbarkeit des Quellcodes[6] wieder ohne eigenes Training des Modells keine neuen Stücke erzeugt werden können. Zudem ist die Qualität der *MuseGAN*-Stücke aufgrund der enormen Komplexität des selbstgestellten Problems nach Ansicht des Autors dieser Arbeit derzeit noch nicht auf einem sinnvoll bewertbaren Zwischenstand im Kontext von Musikgeschmack angelangt.

Da für alle hier erwähnten System, mit Ausnahme eben des *Computoser*s, keinerlei Geschmacksbewertungen verfügbar sind, sind diese Arbeiten nicht ohne erheblichen Mehraufwand, durch zum Beispiel eigene Bewertungsstudien, in diese Arbeit zu integrieren. Von daher fällt die Wahl im Rahmen dieser Arbeit zunächst einmal auf den *Computoser*.

5.2.2. Funktionsweise

Der *Computoser*-Algorithmus arbeitet auf heuristischen Regelfunktionen, die vorgefertigte Kompositionsentscheidungen auf Basis von vorgefertigten Wahrscheinlichkeitsverteilungen aussuchen. Der Entscheidungsprozess für die nächste Note in einer Melodie ist beispielsweise ein Entscheidungsbaum: Im ersten Schritt wird entschieden, ob das Interval eine Prime (gleicher Ton, 25 %), ein Tonschritt (Sekunde, 48 %), ein Oktavsprung (2 %) oder ein anderer Sprung (25 %) sein soll. Im letzteren Falle bestimmt eine weitere Entscheidung über den Sprung einer Terz (48 %), Quarte (2 %), Quinte (25 %) oder Sexte (25 %). Das selbe Konzept wird analog für die Länge der nächsten Note angewendet [Bozhanov, 2014, Tab.3]. Die Wahrscheinlichkeitswerte sind dabei aus der statistischen Analyse echter Musikstücke abgeleitet. Das System kann aber nicht als HMM definiert

[6] `https://github.com/salu133445/musegan` (zul. abgerufen am 9.6.2021)

werden, da es aufgrund ihrer bedingten Entscheidungskaskaden nicht die Markov-Eigenschaft erfüllt.

Eine Vielzahl an Regelfunktionen findet in einer ebensolchen gewichteten Entscheidungsform Anwendung. Sie bestimmen und beeinflussen Melodiestrukturen, Rhythmus und Metrum, Wiederholungen, Variationen, Fehlerbehandlung von Dissonanzen und schweren Synkopen, Schlusskadenzen, Artikulation und Verzierungen. Der Autor schreibt, dass diese Regeln sich mit Wissen aus fundierten Kompositionslehren, aber ebenso auch aus empirischen Beobachtungen entwickelt haben. Die vollständigen Implementierungen aller Regeln mit Quellcode findet sich online[7].

Die Regeln weisen große Einschränkungen durch ihre starken Vereinfachungen auf. So ist bereits die statistisch unabhängige Analyse von Intervallen und Tonlängen benachbarter Töne für eine tatsächliche Reproduktion von Popmusik unzulänglich. Das ist dem Autor bewusst und er argumentiert, dass er seine Art ein Musikstück zu komponieren algorithmisiert und so einen höchst subjektiven, unbeweisbaren und kaum objektiv evaluierbaren Generator (also eben einen *künstlichen Komponisten*) geschaffen hat. Eine mögliche Sichtweise ist, den *Computoser* als eine komplexe programmierte Partitur zu verstehen. Mithilfe einer Vielzahl an Regeln soll eine hohe Varianz in der Erscheinung der Erzeugnisse hergestellt werden. So kann das partiturhafte Wesen der inhärenten Selbstähnlichkeit vor dem wiederholten Zuhörer hinreichend verdeckt werden.

5.2.3. Kollektiver Musikgeschmack

Da alle über `Computoser.com` verfügbaren Stücke vom selben Generator (eben dem *Computoser*) stammen, muss eine Zuordnungsvorschrift der Stücke zu den zwei Zielmengen $\smile$ und $\frown$ bestimmt werden. Zur Verfügung stehen dazu die Anzahl der *Likes* und *Dislikes* für jedes einzelne Stück. Das macht es zunächst möglich die Stücke nach ihrer Bewertungsdifferenz $\#likes - \#dislikes$ zu ordnen. Diese Größe könnte direktes Ziel der maschinellen Vorhersage sein. Es gibt aber zwei Überlegungen, die dazu führen, dass die ganzzahligen Bewertungen in dieser Arbeit in ein 2-Klassen-Problem überführt werden: Erstens ist es nicht sicher, ob die entstehende Rangfolge der Stücke tatsächlich eine vergleichbare Information ist, die einen verallgemeinerbaren Lernerfolg bringt. Darauf nicht zu vertrauen schließt diesen Fehler aus. Zweitens sollen Musikstücke, beziehungsweise Merkmale und Strukturen in den Musikstücken automatisiert identifiziert werden, die eben jene zwei Klassen $\smile$ und $\frown$ unterscheiden. Das Ziel ist ein neu-erzeugtes Stück nach „Nichtgefallen“ aussortieren zu können, sodass nur Stücke, die eindeutig der Klasse $\smile$ zugeordnet werden, übrig bleiben.

Das Hauptproblem der Datengrundlage ist, dass die vorliegenden Bewertungen nicht von angeleiteten Probanden oder gar Experten vorgenommen wurden, sondern in einem unkontrollierten und somit nicht verifizierbaren Verfahren entstanden sind. Es soll deshalb eine Teilmenge gefunden werden, die lediglich aussagekräftige Beispiele enthält. Dieser Ansatz ist eine Form der *Training Set Selection* [Cano u. a., 2007]. Lernerfolge auf verrauschten Datensätzen können durch Selektion der Trainingsbeispiele gesteigert und gesteuert werden [vgl. auch Kordos, 2016]. Ein weiterer Vorteil ist die einhergehende Verkleinerung und Vereinfachung der Datensätze, was wiederum die beim KNN-Training nicht unerhebliche Rechenzeit verkürzt.

[7] `https://github.com/glamdring/computoser` (zul. abgerufen am 9.6.2021)

Ein Problembeispiel: Ein Stück hat einen Bewertungswert von 3. Das kann bedeuten, es wurden 3 positive Likes und keine negativen gegeben. Es kann aber auch sein, dass 100 positive *Likes* und 97 *Dislikes* vergeben wurden. Es ist zunächst unklar welcher Fall eine erlernbare Aussage trifft. Daher werden beide Fälle als gleichwertig betrachtet und die Korrektheitswahrscheinlichkeit als der absolute Wert der Bewertungsdifferenz betrachtet.

Deshalb und unter der Bedingung, dass positive und negative Trainingsbeispiele in den Mengen $\smile$ und $\frown$ gleich oft vertreten sein sollen, wird zur Erstellung von Teilmengen folgende Heuristik angewendet: Die Stücke werden nach ihrer Bewertungsdifferenz sortiert. Anschließend werden die k best- sowie die k schlechtest-bewerteten Stücke jeweils den Trainingsklassen $\smile$ und $\frown$ zugeordnet. Je kleiner k, desto geringer wird zwar die Anzahl an verfügbaren Trainingsbeispielen, es gilt aber auch, dass je größer k, desto unsicherer werden die Daten, da mehr Stücke aus der Mitte der sortierten Verteilung in die Trainingsmenge gelangen. Nahe der Mitte haben die Stücke entweder überhaupt keine Bewertungen erhalten oder die Bewertungen gleichen sich nahezu aus. In Abschnitt 6.4 und weiter in 6.4.4 wird gezeigt, dass dieser Ansatz einen Vorteil bringt.

Als bewertete Daten stehen sowohl eine audiobasierte als auch eine symbolische Repräsentation zur Verfügung. Der Computosers geniert seine Resultate als MIDI-Dateien. Diese sind auch auf `Computoser.com` abrufbar und unverändert in die Datenbank übernommen worden. Auf der Website sind zudem generierte Audiodateien im mp3-Format verfügbar. Diese wurden ebenso unverändert übernommen, da die tatsächliche Bewertung auf der Hörerfahrung exakt dieser Dateien durch den Webplayer (siehe Abbildung 5.2) und den entsprechenden subjektiven Klangerlebnissen beruht.

6. Experimente

Mithilfe der zwei Datenbanken, die im vorherigen Kapitel vorgestellt wurden, sollen nun verschiedene KNN-Architekturen und verschiedene Repräsentationen auf ihren Erfolg beim Versuch der Geschmacksklassifikation überprüft werden. Eine detaillierte Vorstellung und Parameterdefinitionen der gewählten Repräsentationen folgen in Abschnitt 6.1. Die Architekturtypen und ihre konkreten Umsetzungsvarianten werden in Abschnitt 6.2 präsentiert. Dann folgt nacheinander die Auswertung auf dem Datensatz für *DeepBach* (Abschnitt 6.3) und für *Computoser* (Abschnitt 6.4). Da wenige nutzbare Erkenntnisse auf dem Gebiet vorhanden sind, gibt es im Bezug auf KNNs eine enorme Menge an Parametern, die optimiert werden könnten. Zur Beschränkung mussten viele Parameter mithilfe empirischer Begründungen im Vorhinein festgelegt werden. Später werden dann einige Parameter zur Behebung festgestellter Problematiken im Laufe der Experimentreihen weiter optimiert. So fließen Erkenntnisse aus den *DeepBach*-Experimenten in den Aufbau der anschließenden Experimente auf dem *Computoser*-Datensatz ein. Die Praxis Parameter im Laufe von Experimentreihen empirisch anzupassen ist typisches Vorgehen auf dem Gebiet der KNN-Optimierung und wird generell empfohlen [vgl. Goodfellow u. a., 2016, S.421f]. Automatische Optimierung ist derzeit noch sehr aufwändig und wird in den Experimenten in geringem Maße genutzt. Ihr Potential wird im Folgenden in verschiedenen Kontexten diskutiert.

Zur Verifizierung der Modelle wurde eine 5-fache Kreuzvalidierung mit Validierungs- und Testmenge im Verhältnis 3:1:1 gewählt. Die Testmenge dient der Metaverifizierung der Modellwahl durch Anwendung des *Early-Stopping*-Verfahrens mit einer Ausdauer von 5 Epochen und anschließender Wiederherstellung der bis dato besten Gewichtung nach Validierungsfehler (Loss). Als Optimierungsverfahren wird das in Abschnitt 2.3.1 eingeführte *Adam* mit der empfohlenen Lernrate von $\eta = 0{,}001$ genutzt. Implementiert wurden die Experimente mithilfe der Python-basierten Deep-Learning-Bibliothek *Keras* [Chollet u. a., 2015] in der Version 2.2.4 mit *Tensorflow*-Backend [Abadi u. a., 2015] in der Version 2.2.0.

Ein typischer Ansatz zur Verbesserung der generalisierten Trainingsergebnisse ist das *Augmentieren* der Daten im Datensatz [Goodfellow u. a., 2016, S.240f]. Im Kontext dieser Arbeit sind jedoch die Generatoren der Daten bekannt. Vorschnelle Entscheidungen zur Augmentierung können das eigenständige Training der flexiblen KNNs beeinflussen. Ein Beispiel: Typische Augmentierung bei MIDI-Daten ist die Überführung in eine einheitliche Tonart oder gar die Vervielfältigung durch Transponieren in alle zwölf Tonarten. Gemeinhin wird angenommen, dass der Tonartwechsel keinen Einfluss auf die Wahrnehmung eines Stückes hat. Das stimmt vermutlich auch für die meisten üblichen Anwendungsfälle. In diesem Fall könnte es aber sein, dass der *künstliche Komponist* aufgrund seiner Programmierung unbeabsichtigt in gewissen Tonarten bessere oder schlechtere Stücke generiert, sodass die Tonart hier zu einem sinnvollen Entscheidungskriterium werden kann.

Um solche Einflüsse gänzlich auszuschließen wird auf Augmentierung im Rahmen dieser Arbeit verzichtet. Selbiges gilt für normalerweise sinnvolle Vorverarbeitungsmaßnahmen wie der Normalisierung der Daten durch zum Beispiel eine Z-Wert-, Min-Max- oder Pro-Kanal-Normalisierung. Die Daten weisen aufgrund ihrer gemeinsamen Herkunft bereits eine hohe Kohärenz auf, deren Abweichungen eben die gesuchten Entscheidungskriterien sind. So würde eine Normalisierung der Daten eher wichtige Informationen verdecken.

Die Größe der Batches ist variabel: Ein Batch wird von Beispielausschnitten aus stets einem einzigen Stück generiert. So berechnen sich die einzelnen Gradienten in gleichem Maße für alle Stücke unabhängig von deren Länge. Längere Musikstücke führen dabei zu größeren Batches. Im *DeepBach*-Datensatz gibt es Stücke mit einer Länge von 13 s bis 97 s. Im Mittel sind es knapp 28 s mit einem Median von 24 s. Das ist eine akzeptable Spanne. Beim Computoser gibt es Stücke von 14 s bis über 8 min bei einem Median von knapp 105 s. Bei Letzterem wird deshalb eine Länge von maximal 2 min berücksichtigt. Empirisch wird angenommen, dass die Geschmacksentscheidung der vorliegenden *Likes* und *Dislikes* spätestens innerhalb dieser Zeitspanne getroffen wurde. Vermutlich standen die meisten Geschmacksentscheidungen sogar eher deutlich früher fest.

6.1. Spezifikation der Repräsentationen

Von den in Abschnitt 2.1 vorgestellten Repräsentationen wurden das Audiosignal-basierte Mel-Spektrogramm sowie die MFCCs und die MIDI-basierenden symbolischen Repräsentationen Pianoroll sowie MidiCSV gewählt. Das pure Audiosignal (Waveform) wird als Ausgangsrepräsentation für die beiden erstgenannten im Folgenden ebenso vorgestellt. Eine direkte Klassifikation auf der Waveform zeigte in Vorversuchen keinen Erfolg. Aufgrund der Komplexität des Ansatzes (auch in puncto Hardware) wurde deshalb im Rahmen dieser Arbeit darauf verzichtet.

Als Eingabe in die KNNs wurden die Stücke der Datensätze in Ausschnitte fester Länge mit einer festen Schrittweite von 1 s zerstückelt. Die Ausschnittsbreiten werden in den folgenden Abschnitten im Rahmen der fortlaufenden Auswertung der Experimentreihen diskutiert, evaluiert und angepasst (vor allem in Abschnitt 6.3.4).

Waveform Das pure Audiosignal ist eine Wellenform und wird im Folgenden mit *Waveform* bezeichnet. Die vorliegenden Daten werden zur Vereinfachung in ein einkanaliges Monosignal überführt. Zusätzlich wird *Downsampling* auf 22,5 kHz durchgeführt. Ein Eintrag im Eingabevektor entspricht somit einer Zeitspanne von 0,045 ms. Die höchste in der Analyse durch FFT zur Verfügung stehende Frequenz ist somit $f_{max} = \frac{22{,}5\,\text{kHz}}{2} = 11{,}25\,\text{kHz}$ (Nyquist-Frequenz, vgl. Butz [2015, S.102]). Diese und folgende Audioverarbeitungsschritte sind mithilfe des Python-Toolkits *Librosa* [McFee u. a., 2020] in der Version 0.8.0 implementiert worden.

MelSpec Die Umwandlung in ein Mel-Spektogramm (im Folgenden kurz als *MelSpec* bezeichnet) basiert auf einer STFT der Waveform mit 2048 Samples pro Fenster und einer Schrittweite von 512 Samples. Das entspricht einer Überlappung von 75 %. Diese Werte wurden mithilfe der Mel-Skala in 128 Mel-Werte überführt (vgl. Gleichung (2.3)). Ein Eintrag im Eingabevektor entspricht einer Zeitspanne von 23,22 ms.

MFCC Die Umwandlung in MFCCs nutzt dieselbe Fensterung wie MelSpec. Die Anzahl der Koeffizienten wurde mit 20 höher als die typischen 13 gewählt. Es wird vermutet, dass die KNNs die Entscheidung darüber, welche Koeffizienten aussagekräftig sind, selbständig treffen werden. Die Untersuchung dieser Vermutung erfolgt in Abschnitt 7.3.2.

Pianoroll Die Pianoroll ordnet die MIDI-Events in ein zweidimensionales Diagramm mit den Achsen Zeit und Tonhöhe ein. Da das Tempo der Stücke das potentielle Gefallen beeinflusst, geschieht diese Transformation nicht in Relation zum Metrum, sondern durch zeitliche Verrasterung des MIDI-Streams. Da NoteOn- und NoteOff-Events nach Definition einen beliebigen Zeitpunkt besitzen können, werden die Zeitschritte mit 10 ms hinreichend klein gewählt.

Im Falle von komplexen Kompositionen mit mehreren Instrumenten (so der Fall beim *Computoser*-Datensatz) wird die Anschlagsstärke der aktuell klingenden Töne summiert. Eine Variante wäre die gestapelte oder dreidimensionale Transformation unter Berücksichtigung der MIDI-Kanäle. Zum einen vergrößert dies jedoch den Eingabevektor um den Faktor 16. Zum anderen beinhalten die Kanäle einer MIDI-Datei nicht notwendigerweise eine trennende Aussage. So kann nicht ausgeschlossen werden, dass eine zusammenhängende Melodie zwischen den Kanälen springt.

PianoDrumroll Einen entscheidenden Unterschied macht hingegen der MIDI-Kanal 10. Dieser ist als einziger Kanal den perkussiven Klängen wie Schlagzeug oder Perkussionsinstrumenten, also Klängen ohne Tonhöheninformation, vorbehalten. Die hier vorgeschlagene PianoDrumroll stapelt deshalb die Pianoroll der anderen 15 Kanäle wie im vorherigen Unterabschnitt geschildert mit einer Matrix der Events auf Kanal 10 übereinander. Erfolgreiche Anwendung einer Trennung von harmonischen und perkussiven Elementen zur Klassifikation von symbolischen Musikdaten findet sich beispielsweise in Armentano u. a. [2017].

MidiCSV `MIDICSV`[1] ist ein Kommandozeilenprogramm zur Überführung von binären MIDI-Streams in lesbaren Text. Dieser Text kann durch KNNs, die zur Textklassifikation entworfen wurden, verarbeitet werden. Dies geschieht typischerweise in sequentieller Analyse durch LSTMs [vgl. z.B. Vinyals u. a., 2015]. Das Problem wird vereinfacht, indem die Redundanzen durch Kodierung aufgelöst werden. Der MIDI-Stream als Abfolge von NoteOn- und NoteOff-Events wird im Kontext dieser Arbeit in eine Liste von Einträgen mit den vier Informationen Zeitstempel, Tonhöhe, Anschlagsstärke und Midi-Kanal überführt. Aufgrund der variablen Länge des Eingabevektors kann diese Repräsentation im Rahmen der Experimentreihe nur sinnvoll in den LSTMs verarbeitet werden.

6.2. Spezifikation der Architekturen

Als Architekturen sollen in den Experimenten die in Abschnitt 2.2 identifizierten Grundtypen MLP, CNN und LSTM Anwendung finden. Klassisches Problem ist dabei die Optimierung der Architektur auf den konkreten Anwendungsfall anhand der Parameter, die Breite, Tiefe und

[1] `https://www.fourmilab.ch/webtools/midicsv/` (zul. abgerufen am 9.6.2021)

Schichtreihenfolge bestimmen. In dieser Arbeit ist der Anwendungsfall die Modellierung des Musikgeschmacks wie in Abschnitt 1.1 eingeführt. Dabei soll zunächst herausgefunden werden, welche Strukturtypen sich für die unterschiedlichen Repräsentationen aus dem vorherigen Abschnitt am besten eignen. Im Folgenden werden einige Architekturen vorgeschlagen, die aus verwandten Arbeiten abgeleitet und im Hinblick auf vermutete Vergleichshypothesen modifiziert sind. Zudem stellt sich die Frage, ob bestimmte Architekturen generell und nicht nur auf bestimmten Datensätzen erfolgreicher sind. Deshalb werden alle Architekturen in unveränderter Form einmal auf den *DeepBach*- und einmal auf den *Computoser*-Datensatz angewendet, um vergleichende Rückschlüsse zu ermöglichen.

Für alle folgenden Architekturdefinitionen gilt: Die Eingabeschicht wird jeweils an die genutzte Repräsentation angepasst. Die Zwischenschichten nutzen ausnahmslos die etablierte ReLU-Funktion als Aktivierung. Die Ausgabeschichten (jeweils der unterste Eintrag in den Tabellen) nutzen bei den LSTMs die Sigmoid-Funktion und ansonsten Softmax als Aktivierung. Die Schichttypenbezeichnungen decken sich mit den in Abschnitt 2.2.5 Eingeführten.

MLP-Varianten Es werden vier verschiedene MLP-Architekturen festgelegt, deren Konstruktionen in den Tabellen 6.1a bis 6.1d dargestellt sind. Die Bezeichnungen der Architekturen MLP1, MLP2 und MLP3 weisen auf die Anzahl der verdeckten Schichten hin. Die Anzahlen wurden so gewählt, um zu testen, ob und welchen Einfluss die Tiefe des Netzes auf die Trainierbarkeit und den Trainingserfolg hat. Als Ausgangsgröße wurde die doppelte Höhe der Pianoroll und des MelSpec gewählt. Das entspricht der Höhe der PianoDrumroll und dem mehr als 7-fachen der MFCCs. Es wird davon ausgegangen, dass so alle Repräsentationen einen Trainingserfolg erreichen können. Aus Vergleichsgründen wird unabhängig von der Eingangsgröße immer dieselbe Verschaltung und dieselbe Zahl an Neuronen gewählt. Zusätzlich wird das in Relation kleine Vergleichsnetz MiniMLP hinzugefügt. Dieses soll die Annahme verifizieren, dass eine geringe Tiefe und Breite die Ergebnisse deutlich verschlechtert.

Vorerfahrungen aus verwandten Arbeiten können hier (im Gegensatz zu CNNs und LSTMs) kaum genutzt werden. MLPs können je nach Anwendungsfall sehr unterschiedlich beschaffen sein. Besonders bei MLPs bietet sich eine automatische Optimierung der Netze an, die aufgrund des hohen Aufwands an sich innerhalb dieser Arbeit nicht umgesetzt wurde. Eine verbreitete Heuristik ist die Halbierung der Neuronenanzahl mit jeder Schicht als Kanalisierung des Infor-

Typ	Param.
Flatten	
Dense	32
Dropout	0,5
Dense	2

(a) MiniMLP

Typ	Param.
Flatten	
Dense	256
Dropout	0,5
Dense	2

(b) MLP1

Typ	Param.
Flatten	
Dense	256
Dropout	0,5
Dense	128
Dropout	0,5
Dense	2

(c) MLP2

Typ	Param.
Flatten	
Dense	256
Dropout	0,5
Dense	128
Dropout	0,5
Dense	64
Dropout	0,5
Dense	2

(d) MLP3

Tabelle 6.1.: Vorgeschlagene MLP-Varianten im Vergleich.

mationsflusses zur meist kleinen Ausgabeschicht. Hier sind es am Schluss zwei Neuronen für die beiden Klassen ⌣ und ⌢. Darüberhinaus wird nach jeder Neuronenschicht die Eingabe in die nächste mit einem Dropout von 0,5 versetzt. Das ist typisches Vorgehen, um das Netz zu Regularisieren, um eine rapide Überanpassung zu verhindern.

CNN-Varianten Für den äußerst populären und erfolgreichen Forschungsansatz CNN gibt es eine Menge an verwandten Arbeiten, aus denen Architekturvorschläge übernommen werden können. Ein in den letzten Jahren erfolgreiches, ursprünglich für Bilder konzipiertes und später erfolgreich auf verschiedene Anwendungsgebiete übertragenes Netz bezeichnet sich als VGG16 [Simonyan u. Zisserman, 2015]. Es zeichnet sich durch eine enorme Tiefe von eben 16 Schichten aus und ist somit klassischer Vertreter des *DeepLearning*.

Das VGG16 definiert sich durch einen repetitiven Aufbau: Ein Block besteht aus zwei oder drei Conv-Schichten gefolgt von einem MaxPooling. Mehrere solcher Blöcke werden hintereinandergeschaltet. Mit jedem Block verdoppelt sich dabei die Anzahl an Kerneln in den Conv-Schichten. Moderne Varianten bauen dazwischen zusätzliche Dropout-Schichten ein. Diese haben vor den Conv-Schichten eine Ausfallrate von lediglich 0,25, weil die MaxPool-Schichten mit einer Schrittgröße von 2 das Netz bereits zusätzlich regularisieren. Die Größe der Eingabematrizen wird mit jedem Max-Pooling halbiert. Das Netz nimmt so mit zunehmender Tiefe an Breite ab (siehe Trichterform in Abbildung 2.11). Dadurch wird der Informationsgehalt nach und nach reduziert, was ebenso die Chance einer Überanpassung vermindert (vgl. Abschnitt 2.3.3).

Typ	Param.
Conv2D	16 Kernel (3×3)
Conv2D	16 Kernel (3×3)
Flatten	
[Dropout	0,5]
[Dense	128]
Dropout	0,5
Dense	2

(a) MiniCNN[dense]

Typ	Param.
Conv2D	16 Kernel (3×3)
Conv2D	16 Kernel (3×3)
MaxPool	2×2 (Schritt 2)
Dropout	0,25
Conv2D	32 Kernel (3×3)
Conv2D	32 Kernel (3×3)
MaxPool	2×2 (Schritt 2)
Dropout	0,25
Conv2D	64 Kernel (3×3)
Conv2D	64 Kernel (3×3)
MaxPool	2×2 (Schritt 2)
Dropout	0,25
Conv2D	128 Kernel (3×3)
Conv2D	128 Kernel (3×3)
MaxPool	2×2 (Schritt 2)
Flatten	
[Dropout	0,5]
[Dense	128]
Dropout	0,5
Dense	2

(b) DeepCNN[dense]

Tabelle 6.2.: Vorgeschlagene CNN-Varianten im Vergleich.

Das gewählte Netz ist eine Modifikation des klassischen VGG16 und orientiert sich an den Anpassungen für Musikdaten aus Han u. a. [2017] und Bahuleyan [2018]. Der Aufbau besteht, wie in Tabelle 6.2b zu erkennen ist, aus vier der angesprochenen Blöcke. Typischerweise werden im VGG16 an die Conv-Schichten vollvernetzte Dense-Schichten angehängt, um Querverbindungen zwischen den gefalteten Informationspunkten zu ermöglichen. Als Bezeichnung wird deshalb DeepCNNdense gewählt. Als Variante wird dasselbe Netz ohne Dense-Schicht entworfen (in der Darstellung wird die Auslassung mit eckigen Klammern markiert). Dieses wird DeepCNN genannt und soll dazu genutzt werden, später zu überprüfen, ob und welchen Einfluss eine angefügte Dense-Schicht auf den Trainingserfolg hat.

Als Vergleichsnetz wird analog zur Variante MiniMLP ein MiniCNN (siehe Tabelle 6.2a) eingeführt. Dieses hat schlicht einen Block mit zwei Conv-Schichten und verzichtet vollständig auf MaxPooling. Stattdessen folgt direkt die Vernetzung mit der Ausgabeschicht.

LSTM-Varianten Als Vertreter der rekurrenten Architekturtypen werden ausschließlich LSTM-Konstruktionen gewählt. Klassische RNNs ohne internen Zustand waren in Vorversuchen stets weniger erfolgreich und lernten zudem langsamer.

LSTMs entfalten Tiefe durch Informationsrückführung (vgl. Abbildung 2.8) und benötigen deshalb im Gegensatz zu CNNs nur wenige verkettete Module, um effizient zu arbeiten. Li u. Wu [2015, Abb.2] diskutieren und testen verschiedene Kombinationsmöglichkeiten. Die besten Netze sind dort gestapelte LSTMs und LSTMs mit anschließender Dense-Schicht. Deshalb wird an dieser Stelle ein einfaches LSTM (Tabelle 6.3b) und ein doppelt gestapeltes LSTM (Tabelle 6.3c) für die Experimente vorgeschlagen. Beide Netze gibt es zusätzlich mit angefügter Dense-Schicht (LSTMdense und LSTM2dense). Als Vergleichsnetz wird erneut auch ein in Relation kleineres Netz mit der Bezeichnung MiniLSTM (Tabelle 6.3a) eingeführt.

Typ	Param.
Dropout	0,5
LSTM	128
Dropout	0,5
LSTM	32
Dense	2

(a) MiniLSTM

Typ	Param.
Dropout	0,5
LSTM	256
[Dense	128]
Dense	2

(b) LSTM[dense]

Typ	Param.
Dropout	0,5
LSTM	256
Dropout	0,5
LSTM	256
[Dense	128]
Dense	2

(c) LSTM2[dense]

Tabelle 6.3.: Vorgeschlagene LSTM-Varianten im Vergleich.

Der Mensch Darüber hinaus soll für beide Datensätze auch ein Vergleich mit einer händischen Auswertung vollzogen werden. Der Mensch kann hier ebenso als Modell betrachtet werden. Dazu stehen für *DeepBach* begrenzt Evaluationsstatistiken aus Hadjeres u. a. [2017] zur Verfügung. Zusätzlich hat der Autor dieser Arbeit eine eigene Blindbewertung durchgeführt. Der Vorteil dabei: Der Autor hat sich intensiv mit den Stücken auseinandergesetzt. Vor allem mit den Charakteristika von echten Bach-Chorälen gegenüber den gefälschten Chorälen hat er sich während der Erstellung dieser Arbeit und den Datensätzen intensiv auseinandergesetzt. Im Gegensatz

zu den Blindexperimenten aus Hadjeres u. a. [2017], in denen die Probanden lediglich auf ihre Vorerfahrung mit der Materie angewiesen waren, hat der Autor hier konkrete Erfahrungen mit den Datensätzen gesammelt. Das entspricht in gewisser Weise einem Training der Daten ähnlich dem Training der KNNs, die vor dem Test auch Trainingsbeispiele erlernen durften.

6.3. Auswertung: DeepBach

Nun werden alle vorgeschlagenen Architekturen aus dem vorherigen Abschnitt auf dem *DeepBach*-Datensatz getestet. Die echten Bachstücke aus BACH_{369} werden dabei als der gute Geschmack betrachtet. Diese Stücke bilden somit die Klasse $\smile$. Die gefälschten Stücke in FAKE_{369} (vgl. Abschnitt 5.1.3) werden als $\frown$ klassifiziert. Die Stücke werden in unterschiedlichen Tests in den vorgeschlagenen Repräsentationen aus Abschnitt 6.1 behandelt. Dazu wird zunächst eine Zerlegung der Stücke in Ausschnitte von 3 s gewählt. Die Wahl der Ausschnittsbreite wird in Abschnitt 6.3.4 weiter evaluiert und optimiert.

Als Maß für die Güte der Modelle wird zunächst ausschließlich die KKR gewählt. Diese gibt an, für welchen Anteil der Trainingsbeispiele die korrekte Klasse vorhergesagt werden konnte (vgl. Abschnitt 2.3.2). Da die Trainingsbeispiele lediglich Ausschnitte der Musikstücke sind, wird zusätzlich ein angepasstes Maß mit der Bezeichnung KKR* vorgeschlagen. Dieses zählt die korrekten Klassifikationen für komplette Musikstücke. Eine Vorhersage y ist ein Wahrscheinlichkeitswert über die Klassenzugehörigkeit, der eine gewisse Aussage über die Klassifikationssicherheit transportiert. Ein einzelnes Stück wird dann als korrekt klassifiziert betrachtet, wenn das arithmetische Mittel der Vorhersagen aller Ausschnitte aus diesem Stück die korrekte Klasse identifiziert. KKR* optimiert so die Klassifikationrate auf komplette Stücke, da sicherere Ausschnitte mit einem Wert von y nahe 0 beziehungsweise 1 durch die Mittelung stärker ins Gewicht fallen als unsicherere Vorhersagen mit y nahe 0,5. Die Analyse weiterer Bewertungsmaße erfolgt zum Ende der Evaluation in Abschnitt 6.3.5.

Die Auswertungstabellen in den folgenden Abschnitten präsentieren die KKR und KKR* für die verschiedenen Kombinationen von Netzarchitekturen und Musikdatenrepräsentationen. Um wichtige Ergebnisse visuell hervorzuheben wird folgende Formatierungsvorschrift angewendet: Je nach Art der Tabelle werden die höchsten Werte jeder Spalte oder jeder Reihe **fett** gedruckt. Modelle mit KKR-Werten unter 0,65 werden als „schlecht“ betrachtet und deshalb ausgegraut. Werte unter 0,55 werden als unbrauchbar betrachtet und noch stärker ausgegraut. Die folgenden drei Abschnitte präsentieren die Ergebnisse nach den drei Architekturgrundtypen sortiert. Nach der Optimierung der Ausschnittsbreiten für die bis dato besten Modelle in Abschnitt 6.3.4, wird die Wahl des besten Modells in Abschnitt 6.3.5 diskutiert.

6.3.1. Mehrlagiges Perzeptron

Tabelle 6.4 zeigt die KKR- und KKR*-Werte der MLP-Netze. Jeder Wert setzt sich dabei als das arithmetische Mittel über alle Testmengen nach 5-facher Kreuzvalidierung (vgl. Kapitel 6) zusammen. Dies gilt, wenn nicht anders angegeben, analog für alle Bewertungsmaße.

Zunächst lässt sich grundlegend feststellen, dass MLPs trotz ihrer Einfachheit und derzeit eher geringeren Anwendung in aktueller Forschung einen durchaus beachtlichen Lernerfolg erreichen.

	KKR			KKR*		
	Pianoroll	MelSpec	MFCC	Pianoroll	MelSpec	MFCC
MiniMLP	0,847	0,644	0,505	0,917	0,615	0,503
MLP1	0,863	**0,659**	**0,515**	**0,935**	0,689	**0,552**
MLP2	0,863	0,569	0,506	0,927	0,694	0,533
MLP3	**0,866**	0,554	0,506	0,926	**0,699**	0,500

Tabelle 6.4.: Auswertung der MLPs auf *DeepBach*.

Ein KKR*-Bestwert von 0,935 für die symbolische Pianoroll-Repräsentation mit einem MLP mit nur einer einzigen verdeckten Schicht (im Folgenden kurz Pianoroll-MLP1) ist durchaus unerwartet hoch. Prinzipiell kann man für die Pianoroll-KKR-Werte eine proportionale Verbesserung mit wachsender Tiefe und Breite der MLPs erkennen. Dieser Zusammenhang wird von den KKR*-Werten jedoch nur für MelSpec bestätigt.

Auf den Repräsentationen MelSpec und MFCC fallen die Ergebnisse insgesamt schlechter aus. Diese Repräsentationen weisen komplexere Muster auf, die von MLPs offenbar nicht so erfolgreich gelernt werden konnten. Die MFCCs konnten die Trainingsmenge überhaupt nicht approximieren. Sie kommen über bloßes Raten nicht hinaus. Der KKR*-Bestwert für MelSpec-MLP3 von knapp 0,7 ist hingegen zwar deutlich besser als Raten, bietet aber an dieser Stelle keine zufriedenstellende Klassifikation im Vergleich zur Pianoroll. Darüberhinaus zeigt dieses Modell aber anschaulich, dass der Optimierungsansatz KKR durch KKR* zu verbessern, korrekt ist: Die KKR beträgt vor der Verbesserung für dieses Modell lediglich 0,55, kann also um knapp 15 Prozentpunkte gesteigert werden.

6.3.2. Convolutional Neuronal Network

Tabelle 6.5 zeigt die Auswertung für die CNN-Varianten in analoger Form zur Tabelle aus dem vorherigen Abschnitt. Auch bei den CNNs schneidet die Pianoroll-Repräsentation bei KKR am besten ab. Die Überlegenheit zu den spektralen Repräsentationen ist jedoch marginal. Die KKR* aller drei Repräsentationen zeigen ziemlich ausgeglichene Bestwerte von knapp 0,95. Den insgesamt besten Wert zeigt das Netz MFCC-MiniCNN mit einem KKR*-Wert von 0,951.

Da die Audiorepräsentationen auf immer den selben Orgelklängen basieren (vgl. Abschnitt 5.1.3), sind die guten Werte auf den spektralen Repräsentationen durchaus überraschend. Erwartet worden war eher, dass die Unterscheidung der Klassen mit der Pianoroll auch hier einfacher und erfolgreicher erlernbar ist. Da eigentlich keine Unterschiede in der Klangfarbe zwischen den

	KKR			KKR*		
	Pianoroll	MelSpec	MFCC	Pianoroll	MelSpec	MFCC
MiniCNN	**0,848**	**0,838**	**0,819**	**0,946**	**0,942**	**0,951**
MiniCNNdense	0,701	0,678	0,518	0,711	0,645	0,541
DeepCNN	0,505	0,742	0,508	0,500	0,767	0,554
DeepCNNdense	0,505	0,664	0,505	0,500	0,688	0,500

Tabelle 6.5.: Auswertung der CNNs auf *DeepBach*.

Trainingsbeispielen vorhanden ist, überrascht vor allem der Lernerfolg der auf Klangfarben spezialisierten MFCCs (vgl. Abschnitt 2.1.1).

Die nächste Überraschung ist das schlechtere Abschneiden der DeepCNNs. Für Pianoroll und MFCC waren sie nicht in der Lage die Trainingsdaten zu lernen. Hier müssten weitere Parameter getestet werden, um weitere Aussagen über die Gründe für das Misslingen treffen zu können. Auf MelSpec konnte zumindest ein Lernerfolg erzielt werden.

Ein weiterer Befund ist die stets geringere KKR bei Anfügung einer Dense-Schicht. Für alle Modelle und Repräsentationen gilt, dass bei den Varianten mit Dense-Schicht die KKR im Vergleich zu den Komplementen ohne Dense sinkt.

6.3.3. Long Short-Term Memory

Die LSTM-Varianten sind insgesamt weniger erfolgreich als die CNNs auf allen Repräsentationen und auch als die MLPs auf den Pianorolls. Mit einer KKR* von 0,931 des LSTM2dense auf MelSpec kann der Ansatz in der Spitze aber durchaus mithalten, wie Tabelle 6.6 zeigt.

Im Gegensatz zu den anderen beiden Typen sind die KKR und KKR* der fünf LSTM-Varianten stärker von der Repräsentation abhängig. So konnte Pianoroll am erfolgreichsten vom reinen LSTM gelernt werden, wohingegen MelSpec beim LSTM gerade den geringsten Wert aufweist. Bemerkenswerterweise gilt für LSTM2dense die exakt umgekehrte Aussage. MFCC konnte überhaupt nur vom LSTM2 gelernt werden und dies nur mit minimalem Erfolg. Erstaunlicherweise fällt das KKR* hier sogar um knapp 0,08 schlechter aus. Hier ist die durchschnittliche Klassifikationssicherheit meist nahe 0,5, sodass falsche Sicherheiten sofort zu einer Falschklassifikation führen.

Einen Sonderfall bildet die Repräsentation MidiCSV. Diese kann mit einem KKR*-Bestwert von 0,753 nicht mit den besten Ansätzen mithalten, beweist aber ihre prinzipielle Anwendbarkeit. Problematisch war dabei die reduzierte Trainingsmenge, da bei diesem Ansatz stets ein komplettes Stück als Trainingsbeispiel zugeführt werden muss (vgl. Abschnitt 6.1, deshalb gibt es auch keinen Eintrag für KKR). Somit entfällt auch die direkte Vergleichbarkeit mit den anderen Ansätzen. Die Machbarkeit einer Interpretation von Musikdaten als sequentieller Symbolfluss (ähnlich zur Texterkennung) kann hier bestätigt werden. Das Potential soll aber aufgrund des im Vergleich geringeren Erfolgs in dieser Arbeit nicht weiter ausgelotet werden.

	KKR			KKR*			
	Pianoroll	MelSpec	MFCC	Pianoroll	MelSpec	MFCC	MidiCSV
MiniLSTM	0,823	0,821	0,504	0,882	0,901	0,500	**0,753**
LSTM	**0,825**	0,768	0,534	**0,905**	0,744	0,503	0,674
LSTM2	0,758	0,831	**0,629**	0,797	0,928	**0,550**	0,669
LSTMdense	0,812	0,814	0,534	0,867	0,919	0,500	0,748
LSTM2dense	0,699	**0,832**	0,543	0,680	**0,931**	0,500	0,500

Tabelle 6.6.: Auswertung der LSTMs auf *DeepBach*.

6.3.4. Optimierung der Ausschnittsbreiten

Ein wichtiger Parameter ist die Ausschnittsbreite. Ihre Vergrößerung erhöht sowohl den Rechenaufwand im Training, den Speicheraufwand der Netzgewichte sowie die Komplexität der Information pro Trainingsbeispiel. Daher wurde in den bisherigen Vergleichsexperimenten eine eher kleine Ausschnittsbreite von 3 s gewählt. Der Einfluss der Ausschnittsbreite auf den Klassifikationserfolg soll in diesem Abschnitt beleuchtet werden. Dazu werden die in den vorangegangenen Abschnitten als am erfolgreichsten identifizierten Modelle mit unterschiedlichen Ausschnittsbreiten von 1 s, 5 s, 10 s und 12 s erneut trainiert. Tabelle 6.7 zeigt die tabellarische Gegenüberstellung der Modelle mit Angabe von KKR- und KKR*-Werten. Die gestrichenen Werte für Pianoroll-MiniCNN konnten nicht erhoben werden, da das Modell die Größe des zur Verfügung stehenden GPU-Speichers überstieg.

Zunächst kann der erwartbare Zusammenhang bestätigt werden, dass die KKR mit zunehmender Ausschnittsbreite steigt. Ebenso verhält es sich mit der KKR*. Festzustellen ist dabei, dass die KKR* bei 1 s bis zur KKR* bei 12 s für alle Modelle um nicht mehr als 5 Prozentpunkte steigt, bei KKR hingegen alle eine Steigerung von mindestens 10 Prozentpunkten zeigen. Die besten Modell sind nach KKR stets die mit der höchsten Breite. Bei KKR* finden sich hingegen drei von fünf Modellbestwerten bei einer Breite von 10 s. Das erklärt sich wie folgt: Ohne die Verbesserung durch KKR* sind kleine Ausschnittsbreiten von enormer Schwierigkeit. Mit der Verbesserung durch KKR* können die Falschklassifikationen unter geringer Sicherheit nahe 0,5 durch sicher erkannte Beispiele mit Ausgaben nahe 0 beziehungsweise 1 ausgeglichen werden.

Unter dieser Erkenntnis bestätigt sich auch die Annahme, dass die Wahrscheinlichkeitswerte der Netzausgabe Aussagen über die Sicherheit der Klassifikation beinhalten: Da KKR* stets der Auswertung durch KKR überlegen ist, ergibt das Mitteln der Wahrscheinlichkeitswerte eine Vorhersage mit höherer Sicherheit. Diese Erkenntnis ist entscheidend, um die Durchführung des Abschlussexperiments in Kapitel 8 zu rechtfertigen.

Ein weiterer interessanter Punkt ist der Vergleich der automatischen Klassifikation mit der manuellen Einordnung durch den Autor der Arbeit (Mensch). Dazu wurden für die Breitenkategorien 3 s, 5 s und 10 s jeweils 200 zufällig gewählte Beispiele vorhergesagt. Beim Vergleich der Werte für 3 s und 5 s besteht auch bei der manuellen Klassifikation ein direkter Zusammenhang zwischen Ausschnittsbreite und KKR. Das erklärt sich dadurch, dass mit wachsender Ausschnittsbreite auch die Wahrscheinlichkeit dafür steigt, ein sicher unterscheidbares Merkmal zu entdecken. Das gilt in ähnlicher Form vermutlich ebenso für den Algorithmus. Bei einer

	KKR					KKR*				
	1 s	3 s	5 s	10 s	12 s	1 s	3 s	5 s	10 s	12 s
MelSpec-MiniCNN	0,793	0,838	0,877	0,890	**0,892**	0,923	0,942	0,948	**0,950**	0,939
Pianoroll-MLP1	0,806	0,863	0,880	0,914	**0,918**	0,908	0,935	0,932	0,942	**0,943**
Pianoroll-MLP3	0,816	0,866	0,892	0,926	**0,927**	0,916	0,926	0,930	**0,957**	0,949
Pianoroll-MiniCNN	0,794	0,848	0,875	0,906	—	0,920	0,946	0,949	**0,961**	—
MFCC-MiniCNN	0,756	0,819	0,859	0,888	**0,894**	0,928	0,951	**0,958**	0,943	0,951
Mensch	—	0,630	0,750	0,740	0,690[χ]					

Tabelle 6.7.: Auswertung der bisher besten Modelle mit verschiedenen Ausschnittsbreiten.

Vergrößerung auf 10 s konnte manuell keine Verbesserung mehr erzielt werden, wohingegen der Algorithmus durchschnittlich zumindest geringe Verbesserungen zeigt.

Der Mensch identifiziert vor allem vermeintliche Fehler des *DeepBach* als „schräg“ im Sinne von unpassend dissonant. Eine Evaluierung von *DeepBach* wurde in Hadjeres u. a. [2017] mit Probanden durchgeführt, die in die drei Gruppen Unerfahrene, Musikliebhaber/Musiker und Berufsmusiker/Kompositionsstudenten eingeordnet wurden. Ihnen wurden ebenso zufällige Ausschnitte mit einer Länge von 12 s vorgespielt. Es wurde gezeigt, dass selbst Musik-unerfahrene Probanden ein intuitives Verständnis für solche kompositorischen Fehler besitzen und damit gefälschte Choräle durchaus identifizieren können (KKR = 0,620). Der mit χ gekennzeichnete Wert in Tabelle 6.7 wurde von der dritten professionellen Gruppe übernommen, die wie erwartet auch die besten Ergebnisse erbrachten (KKR = 0,690). Diese Gruppe erzielte einen ähnlichen Werte wie der Autors der vorliegenden Arbeit (KKR = 0,750), wobei letzterer aufgrund der tiefergehenden Beschäftigung mit den Kompositionen (vgl. Abschnitt 6.2) einen kleinen aber offensichtlich messbaren Vorteil hat.

Bemerkenswert ist vor allem die Tatsache, dass die aufgelisteten Modelle selbst bei geringster Ausschnittsbreite bessere Ergebnisse erzielen als der Mensch. Es fällt den KNNs demnach um einiges leichter entlarvende Unterschiede zwischen den Beispielen zu identifizieren. Dies zu erklären wird Teil der interpretierenden Analyse in Kapitel 7 sein.

6.3.5. Das beste Netz für DeepBach

Insgesamt kann man die Experimente auf dem *DeepBach*-Datensatz mit sehr vielen erfolgreichen KNNs, die einen Wert von KKR* > 0,9 erreichen konnten, als gelungen erklären. Vor allem vor dem Hintergrund, dass keine erschöpfende (vollautomatische) individuelle Ausoptimierung der Lernparameter für jedes einzelne Modell durchgeführt wurde. Die Experimente dieses Abschnitts zeigen eine sinnvolle Anwendung der Idee aus Kapitel 4. Sie hinterlassen bis hierhin, auch aufgrund der Komplexität der Fragestellung, bereits viel Raum für weitergehende Forschung.

Die Vermutung, dass Pianoroll als symbolische Repräsentation eine sicherere Vorhersage der Klassenzugehörigkeit ermöglicht, vor allem im Vergleich zu Audioaufnahmen der immerselben Klangfarbe, konnte grundsätzlich nicht bestätigt werden. So kann man MFCC-MiniCNN mit 5-sekündiger und MelSpec-MiniCNN mit 10-sekündiger Ausschnittsbreite aufgrund der geringen Abweichungen als gleichermaßen erfolgreich wie Pianoroll-MiniCNN und Pianoroll-MLP3 mit jeweils 10-sekündiger Breite erklären.

	KKR*	P*	R*
MelSpec-MiniCNN (10 s)	0,950	0,991	0,908
Pianoroll-MLP1 (10 s)	0,942	0,994	0,889
Pianoroll-MLP1 (12 s)	0,943	0,986	0,900
Pianoroll-MLP3 (10 s)	0,957	**1,000**	0,913
Pianoroll-MiniCNN (10 s)	**0,961**	0,984	**0,938**
MFCC-MiniCNN (10 s)	0,943	0,981	0,905
MFCC-MiniCNN (5 s)	0,958	0,983	0,932

Tabelle 6.8.: Vergleich der erfolgreichsten Modelle für *DeepBach*.

Ein weiteres Kriterium soll darüber entscheiden, welches Modell im Abschlussexperiment in Kapitel 8 benutzt wird, um mit *DeepBach* ein möglichst gutes Musikstück hervorzubringen. Unter dieser Zielsetzung wird angenommen, dass es eher hinnehmbar ist, ein potentiell gutes Stück als schlechtes Stück falsch zu klassifizieren (falsch-negativ) und somit zu unrecht auszusortieren, als ein schlechtes Stück als gut anzunehmen (falsch-positiv). Diese Forderung bedeutet statistisch die Forderung nach einem möglichst hohen positiven prädiktiven Wert P (vgl. Abschnitt 2.3.2).

Tabelle 6.8 zeigt für die besten Modelle die jeweiligen P*-Werte. Darin wird nun Pianoroll-MLP3 als optimal zur Nutzung im Abschlussexperiment identifiziert. Obwohl mit einem R*-Wert von 0,913 knapp jedes zehnte gute Stück abgelehnt wird, sind alle der Vorhersagen für die Klasse $\smile$ korrekt. Dies bedeutet, dass das Modell eher zu einer Vorhersage von $\frown$ tendiert, was im Umkehrschluss die Sicherheit für die Klasse $\smile$ erhöht.

Da jeder angegebene Bewertungswert immer der arithmetische Mittelwert der 5-fachen Kreuzvalidierung ist, stehen rein praktisch mehrere Netzgewichtungen zur Verfügung. Aus ihnen wird das Modell gewählt, das mit 0,966 den höchsten Wert für KKR* besitzt.

6.4. Auswertung: Computoser

In diesem Abschnitt sollen die Experimente aus dem vorigen Abschnitt 6.3 analog auf die *Computoser*-Datenbank angewendet werden, um Zusammenhänge und Widersprüche identifizieren zu können. Da es sich nun um komplexere Stücke mit einer unterschiedlichen Anzahl an Stimmen, Klangfarben und Rhythmen handelt, wird zu Beginn bereits eine höhere Ausschnittsbreite von 10 s festgelegt. Diese Wahl stützt sich auf die Erkenntnisse aus Abschnitt 6.3.4. Ansonsten werden die Modelle aus dem vorigen Abschnitt unverändert übernommen, um eine Vergleichbarkeit zwischen den Evaluationen beider Datensätze zu gewährleisten. Als grundlegende Fragestellung muss geklärt werden, ob und unter welchen Umständen sich der kollektive Geschmack generell vorhersagen lässt. Dies wird später durch einen Vergleich des Lernerfolgs des am Ende der Experimentreihe als am besten identifizierten Modells mit einer unabhängigen manuellen Vorhersage durch den Autor der Arbeit beantwortet (siehe Abschnitt 6.4.4).

Eine weitere Fragestellung wurde bereits in Abschnitt 5.2.3 vorbereitet: Es gibt Stücke, die exakt gleich viele *Likes* wie *Dislikes* bekommen haben. Somit ist eine allgemeine Aussage über das Gefallen solch eines Stückes nicht möglich. Stattdessen sollen die in Summe am besten und am schlechtesten bewerteten Stücke als die mit der größten Aussagekraft die Trainingsmenge

	KKR			KKR*		
	250	500	1000	250	500	1000
MelSpec-LSTM2	0,583	**0,603**	0,570	**0,612**	0,604	0,561
MelSpec-MiniCNN	**0,600**	0,568	0,548	**0,628**	0,592	0,565
MFCC-MiniCNN	**0,596**	0,584	0,558	**0,640**	0,620	0,582
Pianoroll-MiniCNN	0,607	**0,618**	0,596	0,624	**0,644**	0,606
Pianoroll-MLP3	**0,621**	0,612	0,591	**0,636**	0,630	0,614
PianoDrumroll-MLP3	**0,624**	0,571	0,602	**0,644**	0,606	0,610

Tabelle 6.9.: Vergleich der bisher besten Netze auf *Computoser*-Trainingsmengen mit 250, 500 und 1000 Stücken.

bilden (vgl. Argumentation zur *Training Set Selection* in Abschnitt 5.2.3). Dies führt allerdings als weiteren Optimierungsparameter die Größe dieser Untermenge ein, der prinzipiell ebenfalls optimiert werden sollte.

Zur ersten Schätzung einer sinnvollen Größe werden die besten Modelle aus den *DeepBach*-Experimenten mit den drei Teilmengen der Größe 250, 500 und 1000 trainiert. Zusätzlich wurde das beste LSTM-Modell MelSpec-LSTM2 hinzugefügt, um ein möglicherweise höheres Erfolgspotential der LSTMs auf der *Computoser*-Datenbank nicht zu übersehen. Tabelle 6.9 zeigt die ersten Ergebnisse für die beiden bekannten Bewertungsmaße KKR und KKR*. Alle in diesem und den folgenden Abschnitten angeführten Tabellen sind weiterhin nach der in Abschnitt 6.3 eingeführten Methodik formatiert.

Zunächst fällt auf, dass die besten Lernerfolge mit KKR* knapp unter 0,65 liegen und im Vergleich zu den Ergebnissen bei *DeepBach* relativ klein ausfallen. Eine weitere Optimierung muss die Zielsetzung der folgenden Abschnitte sein. An dieser Stelle lässt sich aber bereits erkennen, dass sich über alle Modelle hinweg mit einer Beispielmengengröße von 250 bessere Ergebnisse erzielen lassen als mit 500 oder 1000. Einzige Ausnahme bildet Pianoroll-MiniCNN mit einem leicht besseren Ergebnis bei KKR* unter 500.

Für die Auswertung aller Modelle in den folgenden drei Abschnitten wird deshalb eine Größe von 250 gewählt. Die Auswertungen werden analog zu Abschnitt 6.3 erneut getrennt nach Architekturtypen diskutiert. Abschnitt 6.4.4 erläutert eine weitere Optimierung des bis dato besten Modells und den Vergleich mit einer unabhängigen manuellen Vorhersage durch den Autor der Arbeit.

6.4.1. Mehrlagiges Perzeptron

Bei den MLP-Varianten stellen sich prinzipiell wieder MLP1 und MLP3 als die erfolgreichsten heraus. Mit einer KKR* von unter 0,65 kommen die meisten Modelle jedoch nicht weit über bloßes Raten hinaus. Das MLP1-Pianoroll kann mit 0,668 noch das beste Ergebnis erzielen.

MelSpec und MFCC zeigen mit KKR* < 0,55 quasi keinen Lernerfolg. Es bestätigt sich also die Erkenntnis aus der *DeepBach*-Auswertung, dass MLPs für die Pianoroll-Repräsentation am besten geeignet sind. Der Einsatz der erweiterten PianoDrumroll, die perkussive von harmonische Klänge trennt, zeigt keine großartige Verbesserung. Zwar erhält PianoDrumroll-MLP3 knapp den höchsten KKR-Wert, bei KKR* ist die kleinere Darstellung ohne perkussive Klänge jedoch erfolgreicher. Die Nutzung der perkussiven Informationen soll in Kapitel 7 interpretiert werden.

	KKR				KKR*			
	P-Roll	PD-Roll	MelSpec	MFCC	P-Roll	PD-Roll	MelSpec	MFCC
MiniMLP	0,607	0,569	0,503	0,508	0,656	0,584	0,492	0,500
MLP1	0,615	0,596	0,529	**0,523**	**0,668**	0,608	**0,536**	**0,512**
MLP2	0,548	0,561	0,508	0,518	0,588	0,604	0,528	0,484
MLP3	**0,621**	**0,624**	**0,530**	0,510	0,636	**0,644**	0,532	**0,512**

Tabelle 6.10.: Auswertung der MLPs auf *Computoser*.

6.4.2. Convolutional Neuronal Network

Die CNN-Modelle zeigen im Hinblick auf die spektralen Repräsentationen bessere Ergebnisse als die MLPs. So konnte ein gewisser Lernerfolg von KKR* bei knapp 0,65 auf allen Repräsentationen erreicht werden. Die Beobachtung aus der *DeepBach*-Auswertung, dass die tiefen DeepCNN-Varianten auf den vorliegenden Daten keinen Vorteil besitzen, bestätigt sich. Dasselbe gilt für den direkten Vergleich der Varianten mit angefügter Dense-Schicht. Hier sind die Modelle ohne Dense stets erfolgreicher.

Das beste Ergebnis zeigt das PianoDrumroll-MiniCNN mit einer KKR* von 0,648. Ob die PianoDrumroll einen generellen Vorteil gegenüber der einfachen Pianoroll besitzt, kann nur erahnt werden. Insgesamt sind die Klassifikationen durch die getesteten CNN-Modelle in der vorliegenden Form für eine tatsächlich Anwendung eher unbrauchbar.

	KKR				KKR*			
	P-Roll	PD-Roll	MelSpec	MFCC	P-Roll	PD-Roll	MelSpec	MFCC
MiniCNN	**0,607**	**0,635**	**0,600**	**0,596**	**0,624**	**0,648**	**0,628**	**0,640**
MiniCNNdense	0,568	0,559	0,515	0,508	0,584	0,568	0,504	0,500
DeepCNN	0,512	0,496	0,555	0,538	0,500	0,500	0,524	0,504
DeepCNNdense	0,508	0,504	0,553	0,508	0,500	0,508	0,496	0,500

Tabelle 6.11.: Auswertung der CNNs auf *Computoser*.

6.4.3. Long Short-Term Memory

Die LSTMs sind durchschnittlich etwas erfolgreicher als die MLPs und die CNNs. Es kann wie schon bei der *DeepBach*-Auswertung wieder beobachtet werden, dass die Repräsentationen auf unterschiedlichen Architekturen am erfolgreichsten sind. Das heißt, der Erfolg hängt stärker von der richtigen Kombination ab. Auch auf dem Exoten MidiCSV kann mit dem Modell LSTM ein Erfolg von 0,600 erzielt werden, was auch hier eine potentielle Anwendbarkeit stützt.

Insgesamt sind die besten Ergebnisse aber nicht unbedingt besser als bei den anderen beiden Architekturtypen. Es gibt mit dem MelSpec-MiniLSTM allerdings einen positiven Ausreißer mit einer KKR* von 0,708. Dieser Wert ist in einem Bereich, in dem man an eine Weiterverwendung denken kann. Dass das KKR* für MelSpec hingegen antiproportional zur Komplexität des Modells steigt (vgl. Tabelle 6.12), ist als Beobachtung äußerst kontraintuitiv.

	KKR				KKR*				
	P-Roll	PD-Roll	MelSpec	MFCC	P-Roll	PD-Roll	MelSpec	MFCC	MidiCSV
MiniLSTM	**0,608**	0,590	**0,622**	0,537	**0,624**	**0,620**	**0,708**	**0,532**	0,540
LSTM	0,571	**0,591**	0,606	0,523	0,568	**0,620**	0,648	**0,532**	**0,600**
LSTM2	0,546	0,572	0,583	**0,553**	0,576	0,588	0,612	0,528	0,500
LSTMdense	0,529	0,584	0,597	0,526	0,524	0,584	0,604	**0,532**	0,525
LSTM2dense	0,571	0,568	0,563	0,538	0,584	0,552	0,552	0,504	0,530

Tabelle 6.12.: Auswertung der LSTMs auf *Computoser*.

6.4.4. Das beste Netz für Computoser

Die beste Modellkombination ist also das MelSpec-MiniLSTM mit einer KKR* von 0,708. Im Zuge einer weiteren Verbesserung soll, wie bereits ansatzweise in Abschnitt 6.4, die Teilmengengröße der *Training Set Selection* weiter optimiert werden. Die Ergebnisse sind in Tabelle 6.13 angegeben.

Als weitere Testgrößen wurde der Bereich zwischen 200 und 500 in 50er-Schritten abgesucht. 200 wird als absolutes Minimum betrachtet, da darunter die Anzahl der Trainingsbeispiele zum Lernen endgültig zu gering wird. Die Vermutung bestätigt sich, denn 200 konnte keine Verbesserung gegenüber 250 erzielen. Zusätzlich wurden zum Wert 1000 auch 2000 und 3000 getestet. Dabei zeigt sich, dass die mit steigender Trainingsmengengröße immer unsicherer werdenden Urteile tatsächlich die vermutete Verschlechterung der Vorhersage bewirken. Von 500 bis 250 steigt der KKR*-Wert stetig. So konnte der Bestwert von MelSpec-MiniLSTM für KKR* bei 250 für keine Variante der Teilmengengröße verbessert werden.

Als Netzgewichtung für das Abschlussexperiment in Kapitel 8 soll die mit der höchsten Bewertung in den Durchgängen der 5-fachen Kreuzvalidierung gewählt werden. Der maximale Wert ist mit 0,760 noch einmal ein Stück näher an einem zufriedenstellenden Ergebnis. Die KKR*-Werte aller Testsets für das Modell auf Teilmengengröße 250 sind bei einem Minimum von 0,720 mit einem Ausreißer nach unten von 0,560 doch erfreulich hoch. Zudem ist auch der für das Abschlussexperiment wichtige positive prädikative Wert mit $P = 0.694$ der Höchste der Versuchsreihe.

Insgesamt sind die Erfolge auf dem *Computoser*-Datensatz trotzdem eher gering. Nur ein einziges Modell konnte die Geschmacksklasse annähernd zufriedenstellend vorhersagen. Dies liegt mutmaßlich an der streitbaren Datengrundlage, worauf auch der Vergleich mit einer menschlichen Vorhersage hindeutet: Der Autor der Arbeit hat bei einem Vergleichsexperiment selbst versucht die Geschmacksurteile von 150 Stücken durch Anhören der ersten paar Sekunden vorherzusagen. Er kam dabei lediglich auf eine KKR* von 0,669. Er war damit auf Augenhöhe mit vielen der Modellen. Die Aufgabe der Vorhersage kann also an sich durchaus als schwer bezeichnet werden. Das MiniLSTM-MelSpec konnte im Vergleich zur Vergleichsvorhersage durch den Autor tatsächlich sogar besser abschneiden. Es wurden also erkennbare Strukturen durchaus besser gelernt. Diese Strukturen zu identifizieren und ihre Sinnhaftigkeit zu interpretieren ist Hauptanliegen von Kapitel 7.

	MelSpec-MiniLSTM			
	KKR	KKR*	P*	KKR$^*_{max}$
200	0,583	0,625	0,619	0,700
250	**0,622**	**0,708**	0,681	**0,760**
300	0,610	0,667	**0,694**	0,717
350	0,592	0,614	0,584	0,671
400	0,599	0,610	0,592	0,750
450	0,608	0,649	0,628	0,711
500	0,579	0,646	0,641	0,720
1000	0,591	0,618	0,586	0,655
2000	0,562	0,557	0,539	0,655
3000	0,555	0,534	0,520	0,553

Tabelle 6.13.: Optimierung der *Computoser*-Trainingsmengengröße.

7. Analyse und Interpretation

Dieses Kapitel widmet sich der Analyse der im vorangegangenen Kapitel 6 erstellten Netze. Ziel der Interpretation soll es sein, nachvollziehbare Aussagen über die Gründe für die guten beziehungsweise schlechten Vorhersagen der einzelnen Netze zu treffen. Dazu wird zunächst in Abschnitt 7.1 die Interpretation der Neuronenaktivierung der Ausgabeschichten als Kennwert der Qualität definiert und diskutiert. Es werden besonders gute und schlechte Stücke in FAKE$_{369}$ identifiziert, die durch eine in Abschnitt 7.2 vorgeschlagene Methodik sogenannter *Verlaufsplots* weiter verifiziert und untersucht werden. Im letzten Abschnitt 7.3 erfolgt mithilfe der Visualisierungstechniken aus Abschnitt 2.4 eine tiefe Interpretation der Arbeitsweisen der Netze. Im Vordergrund stehen dabei die guten Ergebnisse aus den *DeepBach*-Experimenten, da darauf vielfältigere, weil sicherere Aussagen möglich sind. Hauptaspekt der Untersuchung der *Computoser*-Experimente ist die Erklärung der limitierten Aussagekraft.

7.1. Prinzip der *Bachness*

Wie in Abschnitt 6.3 bei der Einführung des optimierten Vorhersagewertes KKR* diskutiert, kann die Neuronenaktivierung der Ausgabeschicht als Sicherheit der Klassifikation interpretiert werden (vgl. auch Abbildung 2.7a). Die Korrektheit wird im vorliegenden Fall durch die Verbesserung des Klassifikationserfolgs bei Anwendung von KKR* (oder auch P*, R*, etc.) verifiziert. An dieser Stelle wird dieser Faktor mit einer weiteren Interpretation belegt, deren Sinnhaftigkeit und Korrektheit im Folgenden weiter analysiert wird: Die Ausgabe y eines trainierten Netzes für die Klasse $\smile$ soll als Qualitätmaß für die musikalische Güte einer Komposition beziehungsweise eines Musikstücks dienen.[1] Dieser Faktor wird im Folgenden als der *Bachness*-Faktor $\mathcal{B}$ bezeichnet. Je größer $\mathcal{B}$, desto eher handelt es sich bei einem klassifizierten Stück Musik um eine Komposition von J. S. Bach, also um etwas mit dem „Prädikat wertvoll“. Der Faktor $\mathcal{B}$ wird zur Vereinfachung später analog auch für gute *Computoser*-Stücke verwendet.

Das Prinzip der *Bachness* verlangt nach einer Allgemeingültigkeit. So sollten die verschiedenen Netze eine gewisse Übereinstimmung in ihrer Bewertung zeigen. Dabei kann jedoch nur von erfolgreichen Netzen eine sinnvolle Aussage erwartet werden. Um die allgemeine Übereinstimmung und somit die Legitimation für die weitere Nutzung von $\mathcal{B}$ zu untersuchen, werden hier die beiden besten Netze aus Abschnitt 6.3 Pianoroll-MLP3 und Pianoroll-MiniCNN verglichen. Ersteres war am erfolgreichsten und erhält im Folgenden den Index 1, letzteres den Index 2.

Tabelle 7.1 zeigt die Auflistung der vorhergesagten *Bachness*-Werte der beiden Modelle für alle Stücke aus FAKE$_{369}$.[2] Darüber hinaus wird der Rang, also die Position in der sortierten

[1] Klasse $\frown$ ist aufgrund des Zweiklassenproblems stets $1 - y$ von Klasse $\smile$. Deshalb muss dieser Wert nicht gesondert betrachtet werden.

[2] Wichtig zu bemerken: Für jedes Stück wurde hier diejenige Netzgewichtung aus der Kreuzvalidierung genommen, für die sich das vorhergesagte Stück in der Testmenge befand.

id	$\mathcal{B}_1$	$\mathcal{B}_2$	$\Delta\mathcal{B}$	rg_1	rg_2	Δrg
78	0,493	0,477	0,017	1	1	0
106	0,442	0,318	0,124	2	9	7
170	0,429	0,414	0,015	3	2	1
117	0,399	0,337	0,061	4	6	2
94	0,357	0,255	0,101	5	17	12
330	0,352	0,323	0,028	6	8	2
228	0,340	0,398	0,059	7	3	4
48	0,327	0,202	0,125	8	33	25
350	0,325	0,254	0,071	9	18	9
61	0,321	0,317	0,004	10	10	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮

(a) Kopf der nach $\mathcal{B}_1$ sortierten Liste

id	$\mathcal{B}_1$	$\mathcal{B}_2$	$\Delta\mathcal{B}$	rg_1	rg_2	Δrg
⋮	⋮	⋮	⋮	⋮	⋮	⋮
236	0,011	0,036	0,025	360	270	90
338	0,010	0,010	0,001	361	353	8
72	0,010	0,010	0,000	362	354	8
244	0,010	0,005	0,005	363	365	2
181	0,009	0,036	0,027	364	268	96
161	0,006	0,006	0,000	365	364	1
12	0,006	0,007	0,001	366	361	5
51	0,005	0,009	0,004	367	356	11
68	0,001	0,011	0,010	368	350	18
35	0,001	0,004	0,003	369	367	2

(b) Fuß der nach $\mathcal{B}_1$ sortierten Liste

Tabelle 7.1.: Vorhersage der *Bachness* $\mathcal{B}$ von Pianoroll-MLP3 (Modell 1) und Pianoroll-MiniCNN (Modell 2) auf allen Stücken in FAKE$_{369}$. Zusätzliche Informationen sind der jeweilige Rang rg in der sortierten Liste, die Differenzen (Δ) und die $id \in \mathbb{N}^{369}$ des Stücks im Datensatz.

Liste, angegeben. Es fällt auf, dass beide Modelle keines der Stücke falsch klassifizieren ($\mathcal{B}_1$ und $\mathcal{B}_2$ sind stets < 0,5), obwohl Tabelle 6.8 für das Pianoroll-MiniCNN $P^*< 1$ angibt. Das liegt daran, dass die Berechnung von P^* und KKR* hier weiter optimiert wurde, denn die dem Klassifikator pro Stück angebotenen Ausschnitte wurden nicht wie bisher mit einer Schrittlänge von 1 s (vgl. Abschnitt 6.1), sondern um 0,1 s verschoben. Die feinere Schrittlänge wird aufgrund der besseren Ergebnisse in diesem und im folgenden Kapitel genutzt.[3] Alle Stücke aus FAKE$_{369}$ und zugehörige Bewertungen finden sich im MIDI-Format in der digitalen Anlage.

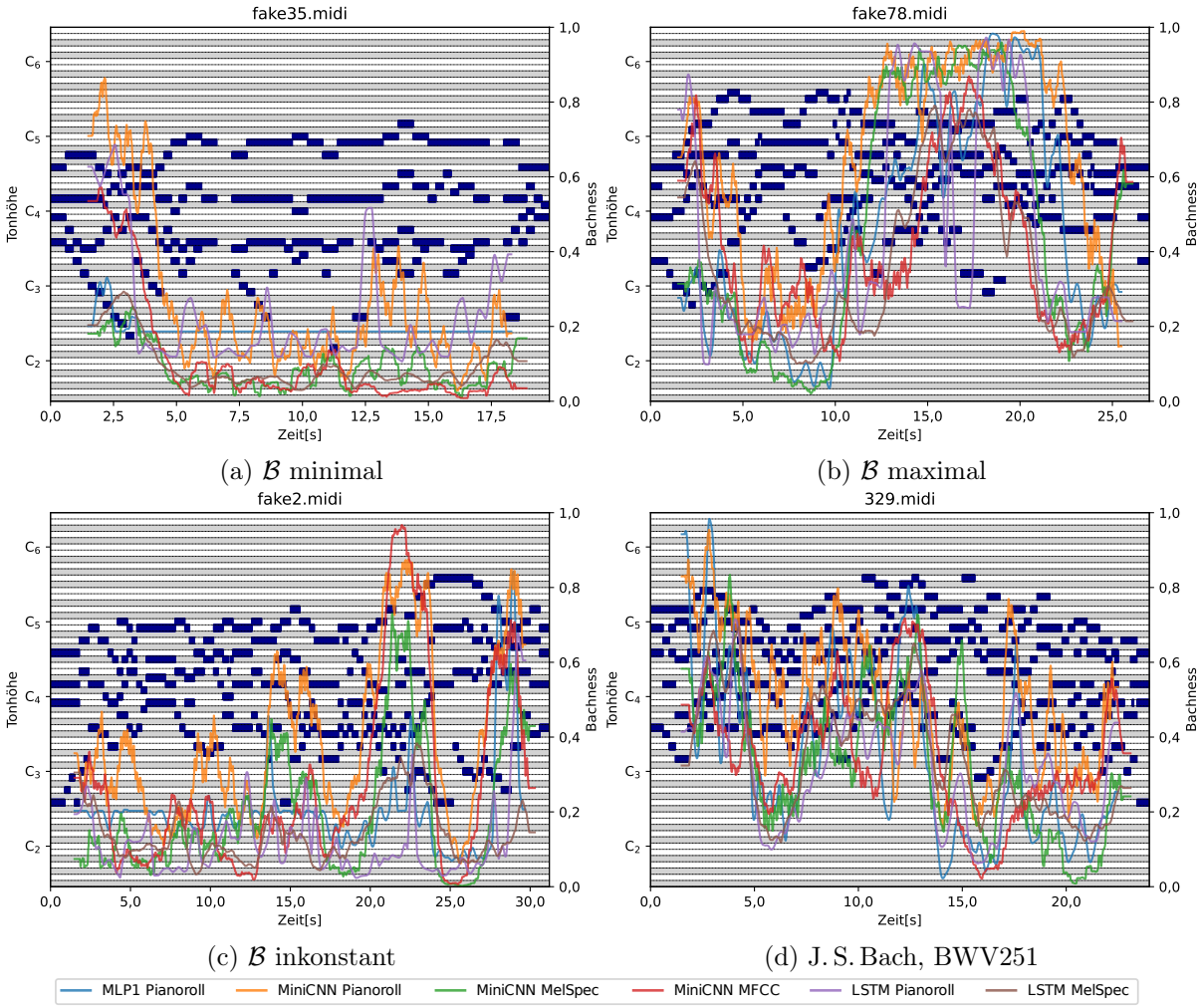
Die Tabelle lässt eine hohe Korrelation zwischen den *Bachness*-Werten der beiden Modelle vermuten. Das arithmetische Mittel der Differenzen $\overline{\Delta\mathcal{B}}$ = 0,038 mit einer Standardabweichung von σ = 0.036 bestärkt diesen Eindruck. Der Korrelationskoeffizient[4] bestätigt mit einem Wert von 0,817 den Zusammenhang. Das zugehörige Korrelationsdiagramm findet sich in Anhang A.1. Selbst der Rang korreliert mit 0,785. Das Prinzip der *Bachness* wird somit als aussagekräftig bestätigt.

7.2. Verlaufsplots

Auf Grundlage des Prinzips der *Bachness* können nun weitere Analysemöglichkeiten entwickelt werden. So werden an dieser Stelle die *Verlaufsplots* vorgeschlagen, die sogar Platz in praktischen Anwendungen finden könnten. Die Idee des *Bachness*-Verlaufsplots ist es, die *Bachness* $\mathcal{B}$ für eine Vielzahl an Ausschnitten zu verschiedenen Zeitpunkten t eines Musikstücks zu generieren

[3] Die Veränderung der Schrittlänge wurde auf Vermutung hin ausprobiert, da mehr Zeit und damit mehr Rechenleistung zur Verfügung stand als für die Vielzahl an Experimenten in Kapitel 6. Es zeigt sich abermals, dass es schlicht zu viele Parameter gibt, die optimiert werden können, als es im Rahmen dieser Arbeit möglich ist. So sind konsequenterweise bessere Netze zu erwarten, wenn auch das Training mit der verkleinerten Schrittlänge wiederholt würde. Das würde jedoch auch den Aufwand pro Trainingsdurchgang um den Faktor 10 erhöhen. Bei der ohnehin sehr viel kostengünstigeren Vorhersage fällt dies nicht so sehr ins Gewicht.

[4] Gemeint ist der Pearsonsche Maßkorrelationskoeffizient: $\mathrm{Korr}(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X)}\sqrt{\mathrm{Var}(Y)}}$



(a) $\mathcal{B}$ minimal (b) $\mathcal{B}$ maximal

(c) $\mathcal{B}$ inkonstant (d) J. S. Bach, BWV251

Abbildung 7.1.: Verlaufsplots der sechs besten Modelle für *DeepBach*. Für die darunterliegende Pianoroll gilt: Töne der schwarzen Tasten eines Klaviers sind grau hinterlegt.

und als $\mathcal{B}$-t-Diagramm darzustellen. Auf einer zweiten y-Achse wird die Pianoroll in der selben Zeitskalierung aufgetragen, um direkte Aussagen in Bezug zu den lokal bewerteten Musikdaten zu ermöglichen. So können gute und schlechte Stellen in den Stücken visualisiert und dann identifiziert werden.

Abbildung 7.1 zeigt die Verlaufsplots für vier ausgesuchte Beispiele, die im Folgenden analysiert werden. Die Analyse erfolgt unter Kenntnis weiterer Plots, die nicht alle Platz in dieser Arbeit finden konnten. Die Plots zeigen Verläufe von $\mathcal{B}$ für die sechs besten Modelle für *DeepBach* aus Abschnitt 6.3 unter Berücksichtigung, dass alle drei Repräsentationen (Pianoroll, MelSpec, MFCCs) sowie alle drei KNN-Typen (MLP, CNN, LSTM) vertreten sind. Dabei wurden die Modelle, die auf Ausschnitten von 3 s trainiert wurden (Abschnitte 6.3.1 bis 6.3.3), ausgewählt, da diese eine lokalere Aussage erlauben. Die Kurven sind auf den ersten Blick relativ unübersichtlich. Sie zeigen dadurch aber gut, dass die meisten Netze äußerst ähnliche Verläufe hervorbringen. So können Ausreißer aus dem Dickicht an Kurven schnell ausgemacht werden. An Ausreißern besteht ein besonderes Interesse.

Abbildung 7.1a zeigt das schlechteste Stück nach $\mathcal{B}_1$ aus Tabelle 7.1. Man erkennt, dass der Anfang als noch relativ gut klassifiziert wird, ab $t = 5\,\text{s}$ dann einstimmig als $\frown$. Für die MelSpec-

Modelle (grün, braun) und das MLP1-Pianoroll (blau) ist $\mathcal{B}$ von Anfang an gering. Die im Konsens schlechte Fortführung der Komposition ist dominiert von langen Noten, die sich oft wiederholen. Es entsteht (auch beim Hören) eine gewisse Stagnation fast wie ein Ostinato. Solch eine Komposition ist für Bach untypisch. Bach ist vor allem für fließende Gegenbewegungen bekannt. Das scheinen die KNNs erkannt zu haben. Zudem sind die entstehenden Akkorde sehr weit. So steht ab $t = 8\,\mathrm{s}$ über dem Bass ein G-Dur-Akkord, allerdings in einer eher untypischen weiten Lage der Form $\{G_3,D_4,B_4\}$.[5] Auch das kann ein entlarvendes Indiz gewesen sein. Die LSTM-Pianoroll (lila) zeigt zudem einen Sprung bei $t = 12{,}5\,\mathrm{s}$. Zusammen mit dem MidiCNN-Pianoroll (orange) wird die folgende Auflösung des Akkords als „möglicherweise eher Bach“ interpretiert.

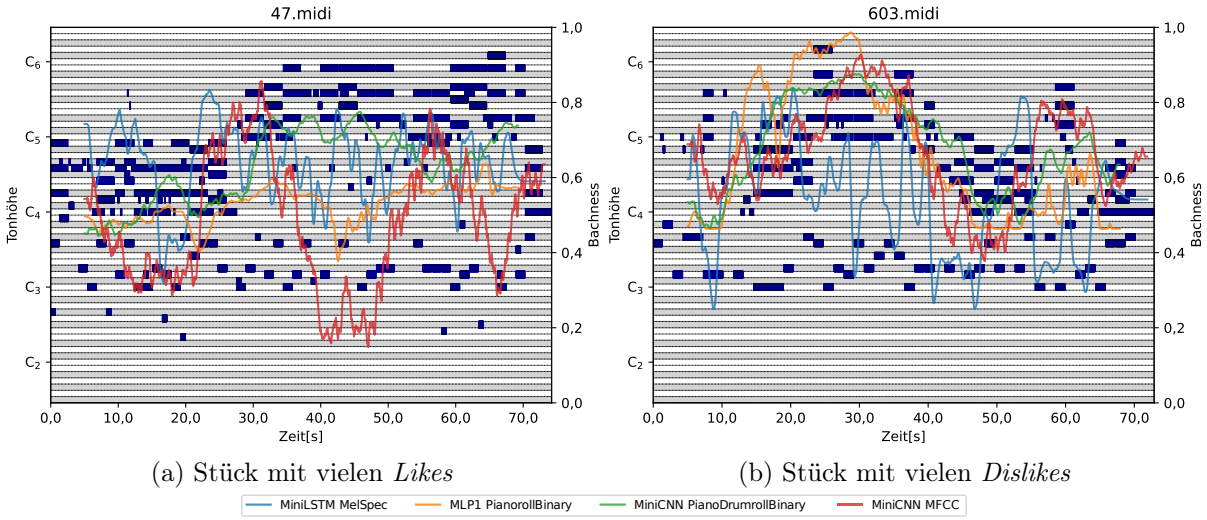
Abbildung 7.1b zeigt als Gegenbeispiel das höchstbewertete Stück nach Tabelle 7.1. Die Werte für $\mathcal{B}$ sind durchweg aber vor allem auf dem Intervall $t \in [13\,\mathrm{s}, 21\,\mathrm{s}]$ eher hoch. Das LSTM-Pianoroll (lila) zeigt bei $t = 17\,\mathrm{s}$ im Gegensatz zu allen anderen wieder einen Sprung nach unten. Vor allem wird die Spanne $t \in [5\,\mathrm{s}, 10\,\mathrm{s}]$ korrekt als ungut klassifiziert. Bei $t = 4\,\mathrm{s}$ steht kurz als Akkord ein A-Dur in der Umkehrung $\{C_{\sharp 4},A_4,E_5\}$, allerdings sehr unschön als kleine None über einem $C_{\natural 3}$. Dies ist selbst für einen Laien als unpassender Moment starker Dissonanz vernehmbar. Danach folgt eine ungeschickte Stimmführung in eine enge Lage, die unvermittelt in eine Weite zurückgeführt wird. Bei $t = 15\,\mathrm{s}$ kann die Stimmführung durchaus als geschickter bezeichnet werden, wie die Verlaufsplots auch richtig implizieren.

Abbildung 7.1c wurde aufgrund des stark schwankenden Verlaufs ausgewählt. So zeigen sich vor allem am Ende zwischen zwei hohen lokalen Maxima ein tiefes konsentes Minimum. Dieses Beispiel zeigt: Die starke melodische Bewegung, die typisch für Bachs Werke ist, wird als positiv erkannt. Die langgezogenen Akkorde, die als Fermaten am Ende einer abgeschlossen Phrase stehen, werden als Bach-untypisch erkannt. Dies kann durch die Studie weiterer Stücke als generelles Phänomen bestätigt werden, obwohl Fermaten zum Phrasenschluss auch bei den Stücken aus $\textsc{Bach}_{369}$ ständig auftreten.

Zum Vergleich der Klassen zeigt Abbildung 7.1d ein echtes Stück von Bach. Dies ist ein Stück, dass fälscherweise mit $\mathcal{B}_1$ und $\mathcal{B}_2 < 0{,}2$ bewertet wurde. Gerade die Stelle bei $t = 16\,\mathrm{s}$ wird als äußerst untypisch gewertet. Die Analyse zeigt: Das ist sie auch. Der Bass hat an der tiefsten Stelle ein B_2 wo die Sopranstimme gerade ein D_5 hat. Das ist ein Intervall einer Dezime plus einer weiteren Oktave. Das kommt bei Bach auch schonmal vor (so auch hier), aber selten wenn die drei Oberstimmen in einer engen Lage eines G-Dur-Akkords in Grundstellung $\{G_4,B_4,D_5\}$ auftreten. Auch im Klangeindruck sind die Oberstimmen getrennt vom Bass zu verorten. Typisch für Bachs Choräle ist jedoch eher eine enge Stimmführung, das heißt eine durchgehend zusammenhängende Struktur aller vier Stimmen. Aber hier gilt, Ausnahmen bestätigen die Regel, was dem automatischen Klassifikator logischerweise eine besondere Schwierigkeit bereitet.

Die musikwissenschaftliche Analyse der Beispiele ist insgesamt bereits sehr ertragreich. So könnte man hier noch viel weiter ins Detail gehen. Auch der intensive Vergleich mit vielen weiteren Plots wäre interessant. Eventuell wäre sogar eine automatische Analyse der Stücke auf musikalische Attribute wie eben Akkorde und Umkehrungen, Tonarten und Tonartswechsel sowie Stimmführungsstrukturen erkenntnisreich. Die Verlaufsplots bieten auf jeden Fall viel Raum für tiefgehende interdisziplinäre Studien.

[5] Notennamen hier und im weiteren werden mit den wissenschaftlichen Namen der C-Dur-Tonleiter C, D, E, F, G, B bezeichnet. Das deutschen „H“ ist das B. Das deutsche „B“ ist ein B♭.



(a) Stück mit vielen *Likes* (b) Stück mit vielen *Dislikes*

Abbildung 7.2.: Verlaufsplots von vier der besseren Modellen für *Computoser*.

Allerdings gilt: Alle in diesem Abschnitt getroffenen Aussagen sind Rückschlüsse von außen auf die Blackbox KNN. Die Vermutungen müssen weiter geprüft werden. Das geschieht im übernächsten Abschnitt 7.3. Aus diesem Abschnitt hervorzuheben ist insbesondere die Erkenntnis, dass die Verläufe der *Bachness* über alle Kombinationen von Architekturtypen und Repräsentationen hinweg, eine große Ähnlichkeit erzielt haben.

Erkenntnisse über die Ergebnisse für Computoser sind ungleich schwerer zu gewinnen. Abbildung 7.2 zeigt zwei Verlaufsplots für einmal das meistgemochte Stück (Abbildung 7.2a) sowie ein eher nicht gemochtes Stück (Abbildung 7.2b). Zu beobachten ist, dass die guten Modelle (vor allem MiniLSTM-MelSpec, vgl. blau in Abbildung 7.2b bei $t = 30\,\mathrm{s}$) ihre eigene Unsicherheit „kennen“, da vor allem Werte um $\mathcal{B} \approx 0{,}5$ auftreten. Das ist für praktische Anwendungen von Vorteil. So kann die generelle Sicherheit mit diesem Feedback (ähnlich zu KKR*) optimiert werden oder es ist zumindest möglich dem potentiellen Nutzer eine Sicherheitsaussage mitzuteilen.

7.3. Tiefe Analyse

In diesem Abschnitt sollen nun die Modelle mithilfe der tiefen Analysemethoden aus Abschnitt 2.4 untersucht werden. Das Hauptproblem ist trotz dieser Visualisierungswerkzeuge weiterhin die Komplexität der Modelle sowie der Problemstellung im Kontext von Musikdaten. So gibt es eine Vielzahl an Schichten, Neuronen und Eingabebeispielen, die man untersuchen und vergleichen kann. Dieser Abschnitt untersucht die aufgeworfenen Fragestellungen aus Kapitel 6 und den bereits identifizierten Phänomenen in den vorherigen Abschnitten. Der Autor hat eine Vielzahl an Visualisierungen durchgeführt, die hier nur ansatzweise besprochen werden können. Nur die interessantesten und aussagekräftigsten Kombinationen von Visualisierungstechniken und Modellkomponenten werden im Folgenden nach Datensätzen und Musikdatenrepräsentationen geordnet präsentiert.

Die Implementierung der Analysemethoden aus Abschnitt 2.4 erfolgt mithilfe der Bibliotheken *tf-keras-vis*[6] in der Version 0.6.1 und *tf-explain*[7] in der Version 0.3.0. Die Versionsnummern weisen darauf hin, wie neu die Entwicklungen auf diesem Gebiet sind. So traten, wie es erwartbar war, praktische Softwareprobleme bei der Durchführung der Analyse auf. So ist auch die Unterrepräsentation von Analysen auf LSTM-Modellen zu erklären, für die rein praktisch weit weniger (funktionstüchtige) Tools zur Verfügung stehen als für die populäreren CNNs. Das ist auch ein Ansatzpunkt für Anschlussarbeiten.

7.3.1. DeepBach und Pianoroll

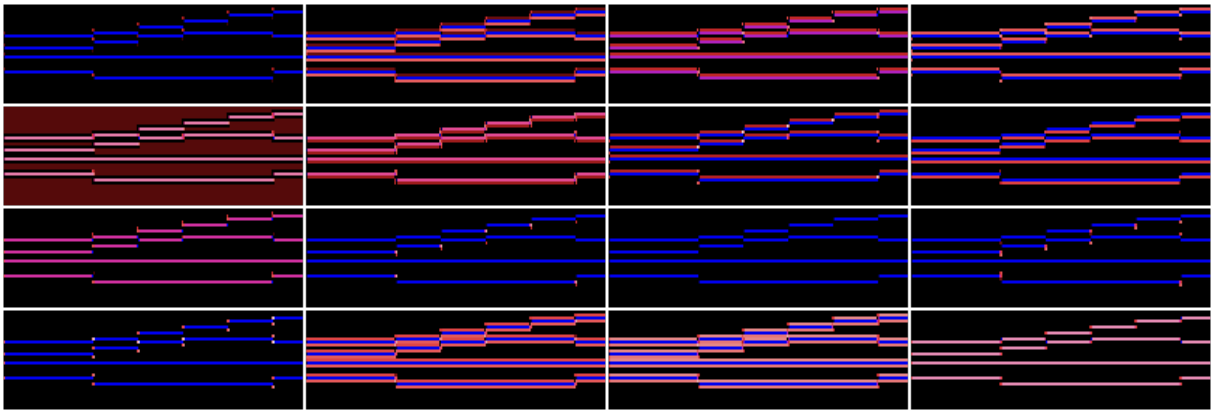
Die erste tiefe Analyse betrachtet das sehr erfolgreiche Netz Pianoroll-MiniCNN. Da es für ein CNN relativ klein ist, können problemlos alle 32 Filter für eine Beispieleingabe dargestellt werden. So zeigt Abbildung 7.3 die Aktivierungen der 16 Filter der ersten Schicht bei Eingabe einer Pianoroll. Die Wahl des Stücks ist in dem Fall nicht so wichtig, da die Kernel bei der Vorhersage die Eingabe immer ähnlich falten. Das Bild visualisiert die Aktivierungsstärke in rot und die Eingabe in blau.

Im Bild wird sichtbar, dass sich verschiedene Arten von Kantendetektionsfiltern ausgeprägt haben. Das war erwartbar bei den begrenzten Belegungsmöglichkeiten eines 3×3-Kernels. Diese können aber nun musiktheoretisch interpretiert werden: So zeigt der obere rechte Filter in Abbildung 7.3a rote Linien exakt eine Reihe oberhalb der Noten. Die Grafik darunter zeigt das Gegenteil, nämlich Linien exakt darunter. Mit diesen Kerneln erhält das Netz in dieser Schicht eine Aussagekraft in Bezug auf gehaltenene kleine Sekunden (Halbtonintervall) im Klangbild. So etwas kommt bei Bach niemals vor, da die kleine Sekunde eine zu starke Reibung für die Musikvorstellungen des Barock mit sich bringt. Somit ist dieser Kernel ein sinnvoller Merkmalsextraktor. Andere Filter dieser ersten Schicht zeigen ähnliche Konstruktionen. So sind mal die beiden umliegenden Halbtöne von Interesse (dritte Spalte, unten) oder gar alles außer den umliegenden Tönen (erste Spalte, zweites).

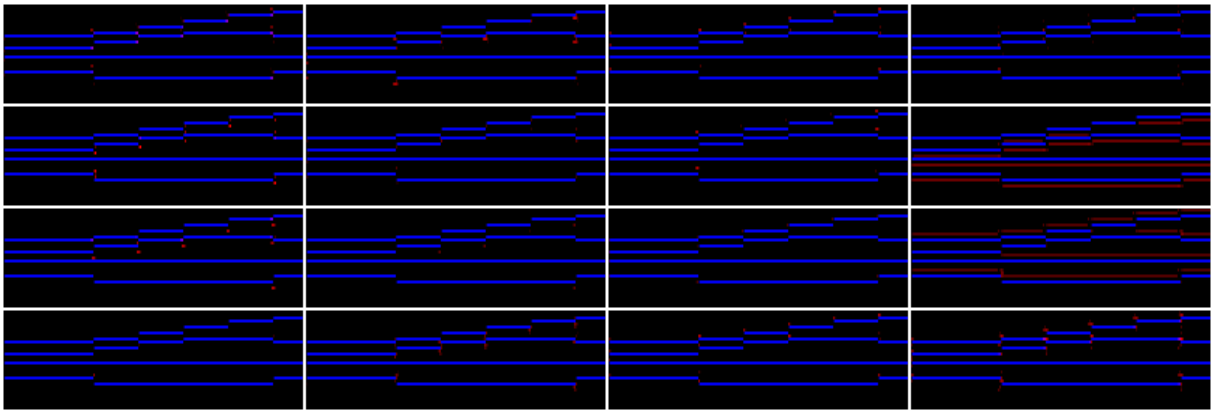
Die anderen Filter zeigen hingegen punktuelles Interesse an den Übergängen. So zeigt der obere linke Filter Interesse am Bereich vor Beginn einer Note einen Halbton darüber. Musiktheoretisch ist das die Frage danach, ob die neue Note über einen Halbtonschritt von oben erreicht wurde. Gerade der Halbtonübergang von unten, der auch in mehreren Filtern auftaucht, ist als Prinzip der Auflösung von Dissonanzen eine wichtige musikalische Qualität [vgl. Kühn, 1987, S.135,195].

Die Filter der zweiten Schicht des Netzes sind in Abbildung 7.3b dargestellt. Sie zeigen die Interpretation der Aktivierungsinformationen der ersten Schicht durch 16 weitere Filter. Es zeigt sich, dass diese Filter nach den selben Prinzipien wie die der ersten Schicht arbeiten und so eine neue Aussageebene betreten. So zeigen die beiden mittleren Filter in der vierten Spalte eine Merkmalsextraktion der Präsenz einer großen Sekunde (Ganztonintervall). Das bedeutet, hier wurde quasi als Verkettung zweier Halbtöne eine Aussage über einen Ganzton hergeleitet. Weiter zeigt sich über alle Bilder, dass die Halbtoninformation oftmals fallengelassen wurde. Der Abstand von zwei Halbtönen (im Bild zwei Pixel) dominiert die Bilder.

[6] `https://github.com/keisen/tf-keras-vis` (zul. abgerufen am 9.6.2021)
[7] `https://github.com/sicara/tf-explain` (zul. abgerufen am 9.6.2021)



(a) Erste Conv-Schicht



(b) Zweite Conv-Schicht

Abbildung 7.3.: Aktivierungen der Filter der einzelnen Schichten des Pianoroll-MiniCNN auf *DeepBach*s schlechtestem Werk Fake35.

Daraus lässt sich eine wichtige Erkenntnis für weiterführende Experimente ableiten: Das MiniCNN ist trotz des guten Erfolgs eigentlich zu klein, um Aussagen über größere Intervalle aus den Filtern abzuleiten. Diese Informationsgewinnung wird hier den Ausgabeneuronen überlassen. Die Einführung einer Dense-Schicht brachte hier trotzdem keine Verbesserung (vgl. Abschnitt 6.3.2). Auch zu tiefe Netze zeigten sich als nicht auf diesen konkreten Fall anwendbar. Das kann auch an der Pooling-Schicht gelegen haben, die schließlich die konkrete Tonhöheninformation verwischt.

Es ist zu vermuten, dass die Aussagekraft der Kernel über Intervalle bei einer Größe von 3×3 mit jeder weiteren Schicht um einen Halbton wächst. Es sollten also künftig durchaus Netze mit mehr als zwei Schichten probiert werden. Eine Alternative bietet zudem auch die Vergrößerung der Kernel. Auf jeden Fall zeigt sich, dass die Kernel auf der Pianoroll musiktheoretische Informationen abbilden können. Zugleich zeigen sich vielzählige Interpretations- und Optimierungsansätze, die in anschließenden Arbeiten weiter verifiziert werden sollten.

7.3.2. DeepBach und Spektrogramme

Auf den Spektrogrammen soll nun eine GradCAM-Analyse (siehe Abschnitt 2.4.1) angewendet werden, um mögliche wichtige Merkmale für die Klassifikationsentscheidungen zu identifizieren. Anders als bei den übersichtlichen Pianoroll-Darstellungen sind die Spektrogramme, die selbst

(a) Fake35 (b) Fake78 (c) BWV251

Abbildung 7.4.: GradCAM von MelSpec-MiniCNN auf *DeepBach*.

(a) Fake35 (b) Fake78 (c) BWV251

Abbildung 7.5.: GradCAM von MFCC-MiniCNN auf *DeepBach*.

bereits eine Abstraktion der eigentlichen Daten darstellen (vgl. Abschnitt 2.4.2), ungleich schwieriger zu interpretieren. Die Analyse der Aktivierungen der Schichten wie im vorherigen Abschnitt brachte auf MelSpec-MiniCNN zunächst keine interpretierbaren Bilder hervor. Sie sind zur Vollständigkeit in Anhang A.2 zu finden. Die einzelnen Filter zeigten zwar ebenso längliche und punktuelle Merkmale, Rückschlüsse auf Musiktheorie oder Klangattribute sind jedoch ohne weiteres nicht möglich.

Deshalb wurde GradCAM als Visualisierungsverfahren gewählt, da es die Möglichkeit einer gemeinsamen Aussage über alle Filter der Schichten bietet. Abbildung 7.4 zeigt die Analyse für drei Stücke: Einmal das am besten (Fake78) und das am schlechtesten bewertete Stücke (Fake35) aus FAKE$_{369}$ (vgl. Tabelle 7.1) sowie das echte Bachstück BWV251 aus Abbildung 7.1d. Auffällig ist, dass stets sehr tiefe Töne die höchsten Aufmerksamkeiten bekommen. Es ist also vornehmlich der Verlauf der Bassstimme und weitere Grundfrequenzen, die die Klassifikation beeinflussen. Die Durchsicht weiterer Bilder bestätigt diese Vermutung. In Abbildung 7.4c sind links im Bild aber auch höhere Frequenzen in geringerem Maße an der Entscheidung beteiligt. Insgesamt waren Obertonstrukturen, vermutlich aufgrund der Vertonung mit einheitlichen Orgelklängen, anscheinend nicht als Entscheidungskriterium brauchbar.

Abbildung 7.5 zeigt die GradCAM-Analyse derselben Stücke aber für das Modell MFCC-MiniCNN. Bei MFCCs wurden in der Vergangenheit typischerweise die ersten Koeffizienten als wichtige Merkmalsträger identifiziert. Das kann bei Durchsicht der GradCAM-Bilder nicht bestätigt werden. Es ist erkennbar, dass gerade Koeffizienten mit einem Index > 13 wichtig für die Klassifikation sind. Da die Klassifikation mit diesem Modell hinreichend erfolgreich war, muss die Herangehensweise an MFCCs im Kontext von Musik überdacht werden. Eventuell würden mehr Koeffizienten zu einer besseren Klassifikation führen. Bei zu vielen Koeffizienten würde sich allerdings die Stärke der MFCCs, eben die weitere Abstrahierung vom Spektralbild, weitestgehend verlieren. Dann wären die MFCCs vom Informationsgehalt nahezu identisch mit der Repräsenta-

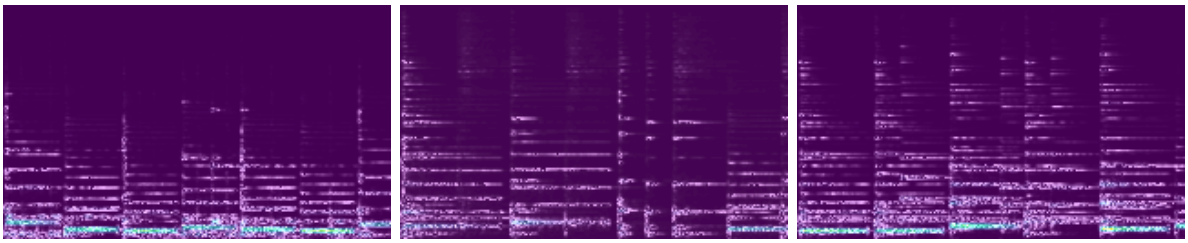
tion der MelSpec. Auch die Repräsentation kann entscheidenden Einfluss auf das Lernen haben (vgl. Kapitel 2), sodass sich eine sichere Einschätzung nur auf Grundlage weiterführender Experimente geben lässt.

Wichtig zu bemerken ist, dass hier tatsächlich das ältere GradCAM und nicht GradCAM++ verwendet wurde. GradCAM erzeugte in diesem konkreten Fall flächigere und damit besser ablesbare Bilder. Dies zeigt noch einmal, dass die für die Bilddomäne entwickelten Ansätze bei der Anwendung auf Musikdaten ihre Aussagekraft nicht unbedingt halten können. Vielmehr sollten Analyse- und Visualisierungstools auf die Audiodomäne angepasst werden. Und auch dort sind Anforderungen an Anwendungen der Musikklassifikation zu anderen Forschungsfeldern wie der Spracherkennung nicht zwangsläufig identisch. Hier ansetzende Arbeiten sollten unbedingt die neue Methode ScoreCAM (siehe Abschnitt 2.4.1) auf Musikdaten testen.

7.3.3. Computoser und Mel-Spektrogramme

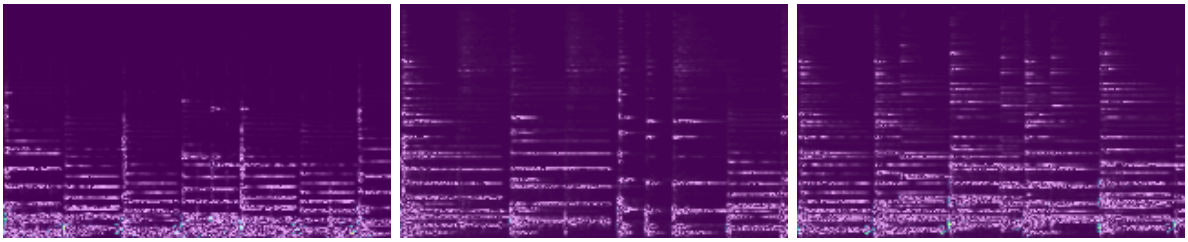
Die Analyse des vorherigen Abschnitts soll nun auf den Computoser-Daten Anwendung finden. Hier war MFCC jedoch weit weniger erfolgreich, sodass im Folgenden nur das Modell MelSpec-MiniCNN betrachtet wird.

Abbildung 7.6 zeigt die Analyse auf dem laut kollektiven Musikgeschmack (siehe Abschnitt 5.2.3) besten Computoser-Stück ($id = 47$) und zwei weiteren zufällig ausgewählten Stücken. Die Beobachtung aus dem vorherigen Abschnitt, dass vor allem tiefe Töne von Interesse sind, bestätigt sich. Abbildung 7.7 zeigt die Analyse auf der zweiten Schicht. Hier ist ähnlich wie bei Abschnitt 7.3.1 die Abstrahierung auf die Tonanfänge, die sogenannten *Onsets*, zu erkennen. Die Schicht erkennt Onsets auf unterschiedlichen Tonhöhen, aber vornehmlich im Grundtonbereich. Klangbildende Obertöne, wie die rauschenden Becken in Abbildung 7.6b (im Spektrum als flächi-



(a) $id = 4$ (b) $id = 47$ (c) $id = 100$

Abbildung 7.6.: GradCAM der ersten Schicht von MelSpec-MiniCNN auf *Computoser*.



(a) $id = 4$ (b) $id = 47$ (c) $id = 100$

Abbildung 7.7.: GradCAM der zweiten Schicht von MelSpec-MiniCNN auf *Computoser*.

ge Schatten oben im Bild zu erkennen), interessieren nicht. Das konnte durch Sichtung weiterer Bilder bestätigt werden.

7.3.4. Computoser und PianoDrumroll

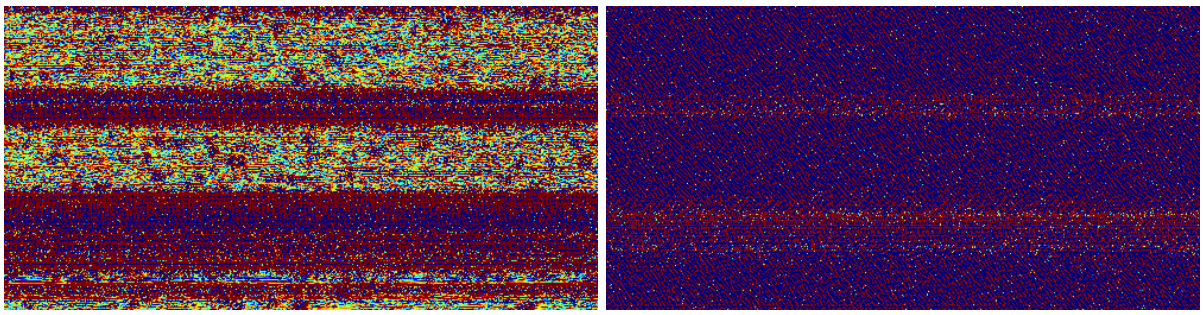
Die PianoDrumroll (siehe Abschnitt 6.1) sollte einen erweiterten Informationsgehalt durch die Trennung von perkussiven und harmonischen Ereignissen ermöglichen. In der Auswertung (vgl. Abschnitt 6.4) konnte höchstens ein geringer Vorteil gegenüber der puren Pianoroll festgestellt werden. Deshalb stellte sich die Frage, ob und inwieweit die Drum- beziehungsweise die Pianoroll wichtig für die Klassifikationsentscheidung war. Die naheliegende GradCAM- und Aktivierungsanalyse brachte keine Besonderheiten hervor. Beide Formen zeigten sich als gleichwertig. Die Bilder der Aktivierungsanalyse gleichen Abbildung 7.3. Die GradCAM (und auch GradCAM++) zeigten lediglich eine gleichmäßige Aktivierung um jede Note und boten damit keine Information zur Beantwortung der Frage. Beide werden deshalb hier ausgespart.

Deshalb wurde ein anderer Ansatz probiert, nämlich der der Aktivierungs-Maximierung (siehe Abschnitt 2.4.2). Die Abbildungen 7.8 und 7.9 zeigen für beide Klassen Ergebnisbilder für jeweils die Modelle PianoDrumroll-MiniCNN und PianoDrumroll-MLP3. Hier ist zunächst einmal im Gegensatz zu den GradCAM-Analysen ein sichtbarer Unterschied zwischen den beiden Klassen erkennbar. So zeigt die Klasse $\smile$ für das MiniCNN deutlich höhere Aktivierungen auch im Bereich der seltener vorkommenden Tonhöhen (gelbliche Bereiche).[8] In den Aktivierungs-Maximierungs-Bildern können also die Bereiche, in denen am häufigsten Noten auftreten, visuell identifiziert werden.

Da die Aktivierungs-Maximierung im Sinne einer Art *Adversarial Attack* quasi die Präferenzen der Netze abbildet, kann eingeschätzt werden, was die Netze für bestimmte Klassen an Eingaben erwarten (vgl. Abschnitt 2.4 und Abbildung 2.15). So sind die Bilder des PianoDrumroll-MiniCNN als stark rauschend zu beschreiben. Die kleinen Kernel führen zur punktuellen Suche von Merkmalen. Beim PianoDrumroll-MLP3 ist das anders. Dort zeigen sich lange horizontale Striche, die das Prinzip von langen Noten beschreiben. Zwischen zwei Noten wird ein Abstand erwartet. Deswegen werden die Striche meist durch den tiefblauen Hintergrund, der für eine Nichtaktivierung steht, getrennt. So zeigt das Modell die Erwartung von Tönen in typischen Intervallabständen. Auch im MiniCNN entstehen in den wichtigen Bereichen diese horizontalen Striche. Etwas tiefere Netze sowie größere Kernel, könnten auch hier potentiell die Strukturen echter Musik genauer abbilden.

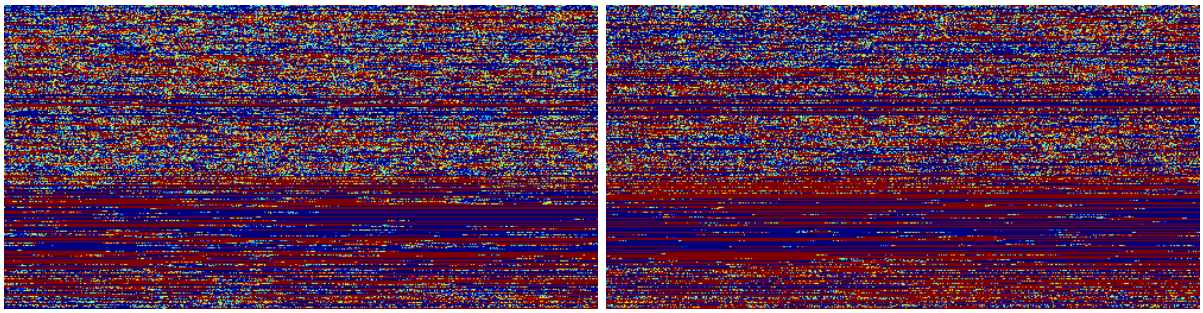
Insgesamt sind die Bilder aber tatsächlich eher verwirrend als hilfreich. Die sonst bei Aktivierungs-Maximierung so erfolgsversprechende Analyse einzelner Schichten und Filter zeigte keine weiteren Interpretationsmöglichkeiten. Strukturen von Musik können nur erahnt werden und sind keinesfalls so eindeutig zu identifizieren wie die Merkmale im Paradebeispiel aus Abbildung 2.15. Unter Umständen benötigen die nicht-bildlichen Daten einfach neue Ansätze.

[8] Für die Drumroll sind das die seltener vorkommenden Instrumente (dünnerer horizontaler rot-blauer Streifen). Auf dem Kanal 10 überträgt der MIDI-Standard die Tonhöhen auf verschiedene Schlaginstrumente. Die meist genutzten Schlagzeug-Sounds liegen dabei im Bereich von 35 (Acoustic Bass Drum) bis 59 (Ride Cymbal 2). Näheres siehe MMA [1991].



(a) Klasse ⌣ (*Like*) (b) Klasse ⌢ (*Dislike*)

Abbildung 7.8.: Aktivierungs-Maximierung auf PianoDrumroll-MiniCNN für *Computoser*.



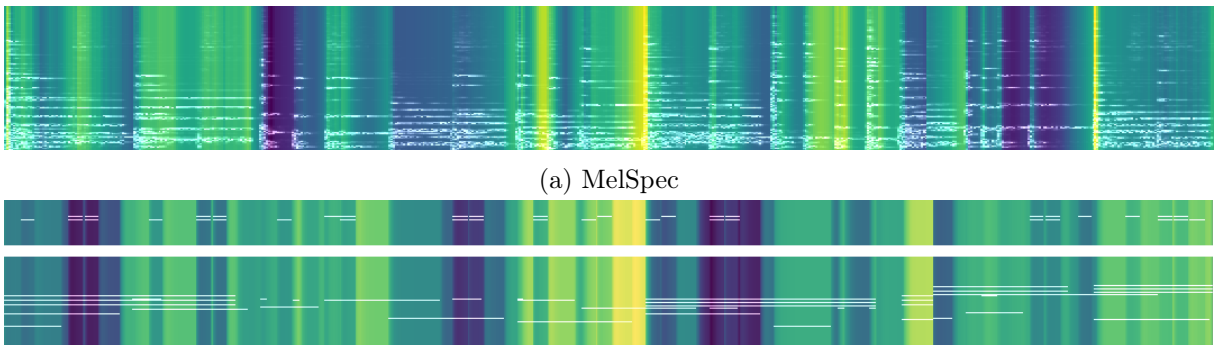
(a) Klasse ⌣ (*Like*) (b) Klasse ⌢ (*Dislike*)

Abbildung 7.9.: Aktivierungs-Maximierung auf PianoDrumroll-MLP3 für *Computoser*.

7.3.5. Computoser und LSTMs

Zum Schluss des Kapitels soll noch das für Computoser beste Modell des MelSpec-MiniLSTMs untersucht werden. Hier gestaltet sich die Interpretierbarkeit aufgrund der noch viel verdeckteren inneren Vorgänge nochmals problematischer. Die Aktivierungs-Maximierung zeigt ähnliche Bilder wie die aus den Abbildungen 7.8 und 7.9.

Deshalb werden in Abbildung 7.10 stattdessen die Aktivierungen der ersten LSTM-Schicht visualisiert. Diese zeigen einen Verlauf über den Ausschnitt, sodass für die LSTM-Zelle interessantere Stellen hellgelb markiert sind. Für MelSpec sind hohe Aktivierungen vor allem bei den Onsets zu erkennen. Bei der PianoDrumroll ist ebenso deutlich, dass über die Länge der Noten die Aktivierung konstant bleibt und sich erst mit der Veränderung durch eine hinzukommende



(a) MelSpec

(b) PianoDrumroll

Abbildung 7.10.: GradCAM des MiniLSTM auf dem *Computoser*-Stück $id = 47$.

oder aufhörende Note die Aktivierung ändert. Die beiden Abbildungen sind in der Zeitachse synchronisiert. Es zeigen sich auch nach Sichtung weiterer Bilder keine interpretierbaren Zusammenhänge, wie zum Beispiel Korrelationen zwischen den synchronisierten Aktivierungsverläufen.

Das Modell ist aber dazu in der Lage, Aussagen über Onsets zu treffen (Auch im Mel-Spektrogramm). Aufgrund des LSTM-Konzepts könnten dadurch prinzipiell Periodizitäten innerhalb der Stücke identifizieren werden. Ansonsten bleibt das LSTM insgesamt mit Abstand die größte Blackbox dieser Untersuchung.

8. Das perfekte Stück

In diesem Kapitel soll nun die tatsächliche Anwendung der KNNs im Sinne des aus Kapitel 4 bekannten *künstlichen Musikproduzenten* durchgeführt werden. Dazu wurden die beiden besten Netze aus Kapitel 6 instrumentalisiert. Für *DeepBach* war es das Pianoroll-MLP3, für *Computoser* das MelSpec-MiniLSTM. Die beiden *künstlichen Komponisten* werden nun darauf angesetzt, pausenlos neue unbekannte Stücke zu generieren. Die KNNs bewerten die Stücke sofort nach deren Erzeugung. Diese Schleife kann theoretisch endlos weiterlaufen. Die beiden folgenden Abschnitte stellen die Ergebnisse für beide Komponisten nacheinander vor.

Die vollständige sortierte Liste und die Stücke im MIDI-Format finden sich in der digitalen Anlage. Die jeweils besten und schlechtesten sind zum Vergleich in einer Audioumsetzung beigefügt.

8.1. DeepBach

Mit *DeepBach* konnten innerhalb der zeitlichen Möglichkeiten dieser Arbeit 1733 Stücke generiert werden. Dabei wurden 1566 mit einem Qualitätswert von $q = 3$ (vgl. Abschnitt 5.1.3) generiert. Im Vorhinein waren die restlichen 167 mit einem zufälligen Wert $q \in [1, 5]$ generiert worden. Der niedrigere Qualitätswert, vor allem nahe 1, zeigte einen Einfluss. So waren die Stücke tatsächlich minderwertiger, da manchmal gar nicht alle der zufällig initialisierten Noteneinträge optimiert wurden (vgl. Abschnitt 5.1.2). Der höhere Qualitätswert hatte jedoch keinen messbaren Einfluss auf die Bewertungsverteilung der Stücke. *DeepBach* zeigte bei $q = 3$ einen ohnehin schon hohen Zeitaufwand von ca. 2 min pro Stück, sodass dieser als Kompromiss gewählt wurde. Die lange Berechnungsdauer lag auch an der Inkompatibilität der Implementierung mit der zur Verfügung stehenden Hardware. So lief der Generator auf nur einem CPU-Kern. Als Länge der Stücke wurde einheitlich der Medianwert der Längen in BACH_{369} mit 24 s gewählt.

Die besten und schlechtesten zehn Stücke zeigt Tabelle 8.1a. Die besten zehn (im Folgenden auch Top10 genannt) wurden alle mit einer Qualität von $q = 3$ erzielt. Abbildung 8.1a zeigt zudem die statistische Verteilung von $\mathcal{B}$. Das gute Stücke seltener auftauchen erkennt man an der zunehmenden Steigung der Kurve. Dasselbe gilt für besonders schlechte Stücke.

Hört man sich die besten Stücke an, so bemerkt man zunächst sehr viel melodische Bewegung in den Stimmen. Das allerbeste Stück (id=01059) kann insgesamt als sehr gelungen bezeichnet werden. Es gibt mehrere überzeugende kontrapunktische Bewegungen (Gegenläufige Melodien). Einzig das Ende wirkt, als wäre das Stück abrupt geschnitten worden. Das liegt an der Unfähigkeit von *DeepBach* wie auch vom *künstlichen Musikproduzenten* überhaupt Form und größere Strukturen des Stückes zu betrachten (vgl. auch Abschnitt 1.4). Das zweitbeste Stück (00237) hat einige harmonische Wagnisse, die Bach so vermutlich nicht komponiert hätte. Es löst seine Dissonanzen mal geschickter, mal weniger geschickt, aber stets auf. Die restlichen Stücke zeigen

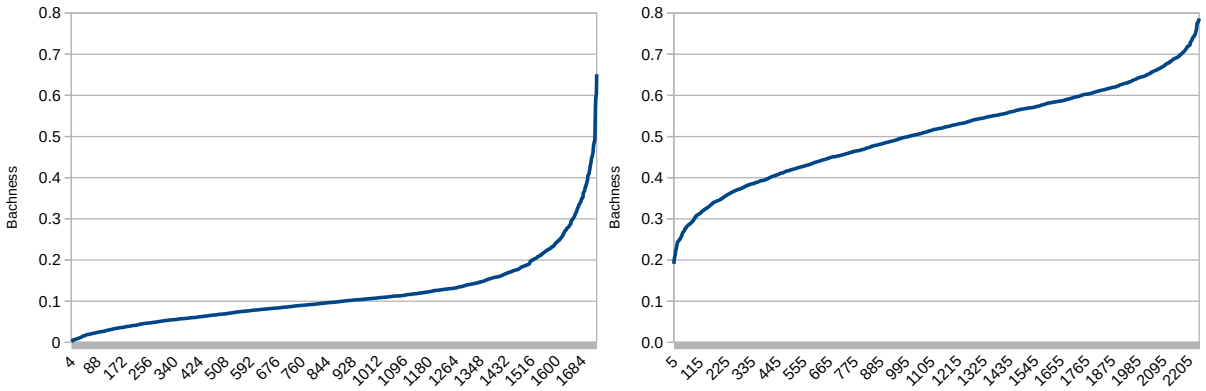
id	*rg*	$\mathcal{B}$	*q*
01059	1	0,651	3
00237	2	0,604	3
00068	3	0,599	3
00123	4	0,593	3
00938	5	0,576	3
01331	6	0,533	3
01433	7	0,494	3
01378	8	0,489	3
00285	9	0,486	3
01297	10	0,484	3
⋮	⋮	⋮	⋮
00525	1724	0,006	3
00774	1725	0,006	3
0087	1726	0,006	1,8
00705	1727	0,006	3
0018	1728	0,005	1,7
0123	1729	0,004	3,3
00663	1730	0,004	3
0082	1731	0,003	3,9
00399	1732	0,003	3
01142	1733	0,002	3

(a) DeepBach

id	*rg*	$\mathcal{B}$
0001970	1	0,787
0001449	2	0,780
0000504	3	0,780
0001312	4	0,779
0000580	5	0,779
0001356	6	0,779
0001914	7	0,776
0001245	8	0,776
0000139	9	0,775
0001528	10	0,774
⋮	⋮	⋮
0000680	2241	0,225
0001552	2242	0,224
0000611	2243	0,218
0001503	2244	0,213
0001180	2245	0,212
0000391	2246	0,208
0000242	2247	0,204
0001757	2248	0,203
0001417	2249	0,199
0001609	2250	0,191

(b) Computoser

Tabelle 8.1.: Liste der besten (Top10) und schlechtesten zehn Stücke nach Einschätzung der *künstlichen Musikproduzenten*.



(a) DeepBach (b) Computoser

Abbildung 8.1.: Statistische Verteilung der Bachness.

ähnliche Charakteristika. Vereinzelt treten auch unschönere Stellen auf. Die Stücke sind aber alle interessant und insgesamt wohlklingend. Das zehntbeste Stück (01297) klingt fast jazzig und bietet nach einer Bearbeitung unter Umständen sogar die Möglichkeit etwas originell Neues aus *DeepBach* hervorzubringen. Der Anfang vom zehntbesten Stück ist als Beispiel für besonders gelungene Stimmführung zu empfehlen.

Die Stücke modulieren[1] insgesamt sehr viel. Bach, der das Spiel mit dem Tonartswechsel sehr gut beherrschte, hätte es aber wohl eher dezenter und gezielter eingesetzt. Zudem werden die Stücke gen Ende schlechter. Zum einen ist dies wohl eine gefühlte Qualität, da man bei Beginn der Stücke erstmal eine Erwartung an den harmonischen Verlauf aufbaut, die dann später durch fehlende Selbstähnlichkeit gebrochen wird. Zum anderen liegt das auch an der zufallsbasierten Methodik der Erzeugung, die zufällige Stellen in der Musik auswählt und immer einzelne Noten im Sinne einer harmonischen Auflösung (d.h. mit größerem Bezug zu den vorangegangenen Noten) manipuliert [vgl. Hadjeres u. a., 2017, Ende Abs.2.2.].

Die schlechtesten Stücke wirken im ersten Moment gar nicht so verkehrt. Das liegt vor allem an der dominanten Nutzung von Moll- und Dur-Dreiklängen, die an sich nie schlecht klingen. Bei genauem Durchhören hört man vor allem wuchtige, repetitive Akkorde (vgl. Stück 00399 oder 0082). Stück 00663 klingt wild und rastlos. Diese Eindrücke rühren von der vermehrten Nutzung von Tonwiederholungen und Parallelbewegungen der Stimmen her. Das ist etwas, das die Top10 Stücke besser gemacht haben, denn Bach war der Pionier einer verwebenden mehrstimmigen Melodieführung.

Hört man noch weiter in die schlechten Stücke hinein, hört man im Vergleich zur Top10 dann doch sehr viele dissonante unstimmige Töne. Unpassende Chromatiken und alterierte Akkorde brechen die Erwartung eines wohlklingenden Bach-Werks. Einige Stellen können nicht anders als mit „schräg“ bezeichnet werden (z.B. 00774). Das ist in diesem Fall ein Erfolg: Der *künstliche Musikproduzenten* hat geholfen die gröbsten Fehler herauszufiltern und so eine sinnvolle interessante Top10 zusammengestellt, die im Gegesatz zu den unteren Zehn ohne richtig schlechte Stellen auskommt. Das beste Stück für sich ist ebenfalls gut gelungen.

Ein weiterer interessanter Aspekt ist das spielerische Wesen der Prozedur. Lässt man sich auf die Personifizierung des *künstlichen Produzenten* ein, kann man einen zusätzlichen Spaß daran entwickeln, die von ihm sortierte Datenbank zu untersuchen und findet so eine Quelle der Inspiration. Und es sei zu bemerken: Die generierten Stücke sind einem Nutzer danach frei zur Weiterverwendung verfügbar. Gute Stellen können bearbeitet oder gar Ausgangspunkt neuer Kompositionen werden.

8.2. Computoser

Mit *Computoser* konnten innerhalb der zeitlichen Möglichkeiten dieser Arbeit 2250 Stücke generiert werden. Dieser *künstliche Komponist* hat eine weit kürzere Generierungsdauer von etwa 10 s bis 15 s. Da das beste Netz hier allerdings auf dem MelSpec arbeitet, kommt zusätzlich die Synthetisierung der Audiodatei und die Transformation in das Spektralbild zum Zeitaufwand hinzu. Somit dauerte die Generierung eines Stücks insgesamt zwischen 20 s und 25 s.

[1] Modulation bedeutet in der Musiktheorie, dass die etablierte Tonart durch eine neue ersetzt wird.

Die besten und schlechtesten zehn Stücke listet Tabelle 8.1b. Abbildung 8.1b zeigt die statistische Verteilung von $\mathcal{B}$. Besonders gute und schlechte Stücke kommen wie zuvor erneut seltener vor. Die Kurve steigt allerdings am positiven Ende weniger stark. Zudem ist der minimale Wert mit knapp 0,2 höher als bei *DeepBach*. Die Aussortierung schlechter Stücke ist demnach weniger konsequent.

Die Top10 der Stücke wirkt auf den unwissenden Hörer zunächst alles andere als wohlklingend. Die Auswahl muss aufgrund des KKR*-Werts von maximal 76 % (vgl. Abschnitt 6.4) als eher unsicherer angenommen werden. Trotzdem gibt es einen klar identifizierbaren Unterschied zwischen den schlechtesten Zehn und den Top10: Die Perkussionsinstrumente zeigen ein interessantes Verhalten. Bis auf das erste Stück enthalten alle einen gleichbleibenden Puls auf einer tiefen Trommel. Die Tempi variieren, aber ein konstanter Trommelrhythmus ist mit Ausnahme des ersten Stückes überall zu hören. Bei den schlechtesten Zehn ist es genau andersherum: Hier hört man auch Rhythmen, darunter auch tiefe Trommeln, aber diese sind meist rhythmisch versetzt und synkopiert oder variieren als Pauke in der Tonhöhe. Der *künstliche Musikkomponist* mag also offenbar tanzbare, gleichbleibende Beats.

Die Harmonik und die Melodien der Stücke sind nicht sonderlich ansprechend. Das liegt aber vor allem auch an dem Generator an sich, der nicht Glanzstück seiner Art, sondern eher durch seine Vorarbeit in puncto Evaluierung hervorgestochen ist (vgl. Abschnitt 5.2.1). Das erste Stück ist eher abstrakte Kunst[2] als ein sinnvolles Musikstück, besonders da sich der *Computoser* als eigenes Ziel die Generierung von Popmusik gesetzt hatte. Der Vorsatz „gute“ Stücke hervorzubringen konnte beim *Computoser* insgesamt eher nicht erreicht werden. Das lag an der dann doch schwierigen Datengrundlage (vgl. Abschnitt 5.2.3) sowie an den qualitativ unzureichenden Erzeugnissen des Generators selbst. Der *künstliche Komponist* scheint sich lediglich auf einen tiefen konstanten Trommelbeat eingeschossen zu haben. Das deckt sich mit der Analyse aus Abschnitt 7.3.3. Dort wurde bereits die Wichtigkeit tiefer Töne durch die tiefe Analyse des Spektrogramms identifiziert.

[2]Der Autor der vorliegenden Arbeit möchte abstrakte Kunst hier nicht abwerten. Im Gegenteil: Er persönlich findet das Stück aufgrund der Qualitäten auf künstlerischer Ebene sogar eher ansprechend.

Teil III.

Fazit

9. Zusammenfassung

Im Zuge dieser Arbeit wurde die Funktion des menschlichen Musikgeschmacks, also die Ordnung von Musikstücken in die Kategorien des Gefallens und Nicht-Gefallens, als binäres Klassifikationsproblem interpretiert. Ein automatischer Klassifikator sollte es einem *künstlichen Komponisten* ermöglichen, seine eigenen Werke zu bewerten und schlechte von guten Resultaten zu unterscheiden. Mithilfe dieses Ansatzes können die Resultate bestehender Musikprogramme nach einer subjektiven Qualität sortiert werden. Er stellt somit die Metaoptimierung eines beliebigen Kompositionsalgorithmus dar. Das Konzept wurde in der *Metapher des künstlichen Musikproduzenten* formalisiert.

Musikgeschmack wurde als ein in der Vergangenheit meist qualitatives Forschungsthema identifiziert. So war die genaue Art der Merkmalsausprägung von Geschmack in Musikdaten vor Beginn der Experimente weitgehend ungewiss. Deshalb wurde das KNN als ein flexibles Lernverfahren hoher Adaptivität gewählt. Die Grundlagen des Verfahrens wurden im Kontext von Musikdaten beleuchtet. Verschiedene Netztypen und Musikdatenrepräsentationen wurden auf ihre Anwendbarkeit untersucht.

Als Datengrundlage wurden zwei unterschiedliche Datensätze aus bestehenden Arbeiten der algorithmischen Komposition abgeleitet. Diese wurden nach einem Literaturvergleich als Vertreter der beiden als grundlegend identifizierten Kategorien der regelbasierten beziehungsweise datenbasierten Systeme gewählt. Der erste Datensatz besteht zur Hälfte aus 369 Choralkompositionen von J. S. Bach (BACH_{369}), die aufgrund ihrer großen Selbstähnlichkeit bereits oft als Kompositionsvorlage in algorithmischen Verfahren der Stilimitation genutzt wurden. Der *künstliche Komponist DeepBach*, der genau für diese Aufgabe implementiert wurde, vervollständigte den Datensatz um weitere 369 sogenannte *gefälschte* Choräle (FAKE_{369}). Lernziel der Klassifikation war es, die echten von den falschen Chorälen zu unterscheiden. Der zweite Datensatz besteht aus Stücken des regelbasierten *Computoser*. Von der Projekt-Website `Computoser.org` standen dieser Arbeit *like* und *dislike* Urteile einer Vielzahl von Besuchern zur Verfügung, aus denen eine Kategorisierung in die binären Geschmackskategorien erfolgte.

Um den angestrebten grundlegenden Überblick über die Anwendbarkeit von KNNs im Kontext von Entscheidungen des Musikgeschmacks zu erhalten, wurde eine Experimentreihe unterschiedlichster Kombinationen von KNN-Architekturen und Musikdatenrepräsentationen gestartet. Die Wahl der Architekturen fiel auf die im ersten Teil der Arbeit als am etabliertesten identifizierten Grundtypen der MLP, CNN und LSTM. Verschiedene konkrete Modellarchitekturen wurden unter Bezugnahme auf verwandte Arbeiten festgelegt. Als Musikdatenrepräsentation wurden die audiobasierten Mel-Spektrogramme sowie die MFCCs gewählt. Zusätzlich wurde die symbolische Notenrepräsentation MIDI einmal als sequentielle Darstellung MidiCSV sowie als Tonhöhen-Zeit-Diagramm in der Form Piano(Drum)roll untersucht.

Die Auswertung auf dem *DeepBach*-Datensatz zeigte direkt einen großen Klassifikationserfolg. Viele Modelle konnten bei einer Fensterung von 3 s bereits KKR-Werte jenseits von 0,8 aufweisen. Die vorgeschlagene Evaluierungsoptimierung KKR* lag sogar für zwölf Modelle über 0,9. Generelle Erkenntnisse waren unter anderem, dass tiefe CNNs im Vergleich zu kleineren Modellen keinen Vorteil zeigten. Ebenso waren Modelle mit angefügter Dense-Schicht generell schlechter. Bei MLPs waren hingegen größere Modelle besser. Die MFCCs boten einen insgesamt geringen Klassifikationserfolg. Als beste Repräsentation stellte sich die Pianoroll heraus. Eine Optimierung der Fensterlänge auf schließlich 10 s wies als erfolgreichstes Modell das Pianoroll-MiniCNN mit KKR* = 0,961 aus. Für den Abschlussversuch wurde aber das Pianoroll-MLP3 gewählt, da es bei einem Wert von KKR* = 0,957 einen Wert von $P = 1$ erreichen konnte. Das bedeutet, kein Stück wurde fälschlicherweise als gutes Stück erkannt.

Der generelle Erfolg für den *Computoser*-Datensatz fiel ungleich geringer aus. Der Datensatz wurde dabei durch Optimierung mit *Training Data Selection* auf die 250 Stücke mit der größten Aussagekraft beschränkt. Es bestätigte sich auch hier, dass tiefe CNNs und angefügte Dense-Schichten den Trainingserfolg verminderten. Die vorgeschlagene PianoDrumroll brachte nur leichte Verbesserungen. Das beste Modell ist hier das MelSpec-MiniLSTM mit KKR* = 0,708 bei einem Fensterausschnitt von 10 s. Der Autor der Arbeit kam in einem kleinen Vergleichexperiment aber ebenfalls nur auf einen Wert von KKR* = 0,669, was den Erfolg positiv relativiert.

Einen besonderen Einblick bot die tiefe Analyse der Modelle mit Hinblick auf mögliche Rückschlüsse über ihre inneren Mechaniken. Die sogenannte *Bachness* $\mathcal{B}$ wurde als Qualitätsfaktor eines Musikstücks vorgeschlagen. Sie leitet sich aus der Neuronenaktivierung der positiven Ausgabeklasse ab. Damit wurde zunächst die Sammlung FAKE_{369} nach Qualität sortiert. Eine Korrelationsanalyse bestätigte das Konzept der *Bachness* auf den zwei besten Modellen für *DeepBach* als hinreichend verallgemeinerbar. Mithilfe von $\mathcal{B}$ konnten Verlaufsplots über die Dauer eines Stücks gezeichnet werden. Die musiktheoretische Interpretation deutet auf einen Zusammenhang zwischen einem positiven Geschmacksurteil und etablierten kompositorischen Strukturen hin.

Anschließend wurde eine tiefe Analyse der Netze mithilfe von *Attention Maps* und Aktivierungs-Maximierung durchgeführt. Es zeigte sich, dass selbst für kleine Modelle wie dem MiniCNN die Untersuchungsansätze quasi unerschöpflich sind. Gerade die Untersuchung der einzelnen CNN-Kernel bot einen interessanten Einblick in die innere Arbeitsweise der Netze. Die gelernten Funktionen konnten mit musiktheoretischen Grundprinzipien in Einklang gebracht werden. So implementiert ein 3×3-Kernel eine Untersuchung der Pianoroll nach Halbtonschrittstrukturen.

Die Schwierigkeiten der tiefen Analyse gerade der spektralen Repräsentationen machte deutlich, dass die hauptsächlich für die Bilderkennung entwickelten Visualisierungsverfahren bei der Betrachtung von Musikdaten an die Grenzen der Interpretierbarkeit stoßen. Das Spektralbild ist anders als ein Foto bereits für sich eine Abstraktion des eigentlich akustischen Objekts. Rückschlüsse sind nur auf geringerem Abstraktionsniveau möglich. Es konnte beispielsweise gezeigt werden, dass vornehmlich tiefe Klänge für die Klassifikation entscheidend waren.

Zum Schluss wurden die Netze in ihrer eigentlich angedachten Funktion als *künstlicher Musikproduzent* eingesetzt. Bei der Prozedur erzeugen *Computoser* und *DeepBach* neue Stücke am laufenden Band, die unmittelbar von den jeweils besten Netzen bewertet werden. Als besonders gut bewertete Stücke entstehen dabei wie erwartet mit einer geringeren Wahrscheinlichkeit als

Mittelmäßige. Der qualitative Vergleich der jeweils schlechtesten Zehn mit der jeweiligen Top10 zeigte prinzipielle Unterschiede. Der *Computoser*, dessen generelle Qualität sich per se auf einem niedrigeren Niveau als *DeepBach* befindet, konnte keine besonders gefälligen Stücke erzeugen. Er zeigte aber eine Vorliebe für repetitive Rhythmen. *DeepBach* zeigte hingegen eindeutig bessere Ergebnisse für die Top10. Die schlechtesten Stücke wiesen klar unerwünschte und vor allem vom Generator ungewollte Dissonanzen auf. Die Top10-Stücke waren allesamt akzeptabel. Bachuntypische Stellen waren vorhanden, zeigten aber nie die Qualitäten eines schlimmen Fehlers. Die angestrebte Metaoptimierung des *künstlichen Komponisten* durch einen *künstlichen Musikproduzenten* kann demnach als erfolgreich bezeichnet werden.

10. Ausblick

Die Grundannahme der rein binären Klassifikation von Musikgeschmack ist durchaus streitbar. Unbestreitbar ist jedoch die große Bandbreite an Erkenntnissen, die sich auf Basis der bewusst beschränkenden Grundprämisse entwickeln konnte. Die Metaoptimierung bestehender Arbeiten zeigte sich im Forschungsbereich der algorithmischen Komposition als durchaus gewinnbringend. Der Autor sieht sich in der Forderung nach Abstraktion von den kreativen Grundprozessen auf größere Zusammenhänge sowie nach der damit einhergehenden Vielfalt an aktiven Gestaltungsprinzipien und Akteuren in ein und demselben Kompositionsprozess bestätigt.

Optimierungsansätze und Ideen für weiterführende Arbeiten waren im Verlauf der Vorliegenden ebenso auffallend wie vielfältig. Zunächst ist da die große Zahl an Parametern, die aufgrund der bereits hohen Komplexität und damit hohen Zeitaufwands der Experimentreihe nicht optimiert wurden. Wie für KNN-Arbeiten typisch mussten viele dieser Parameter bereits vor Durchführung der Experimente festgelegt werden. Den größten Gewinn bringt womöglich eine weitere Optimierung der Netzarchitekturen. Im Rahmen dieser Arbeit wurden diese vor dem Training festgelegt, um einerseits eine Vergleichbarkeit zwischen den beiden Datensätzen herzustellen und andererseits die Einbeziehung aller drei klassischen Netztypen zu ermöglichen. Die automatische Ausoptimierung, klassischerweise mit EAs, bietet sich hier an [vgl. Goodfellow u. a., 2016, Abs.11.4.2]. So wies die tiefe Analyse bereits darauf hin, dass die tiefen CNNs zwar zu tief, das MiniCNN aber womöglich nicht tief genug war (vgl. Abschnitt 7.3.1). So sind auch Schichtkenngrößen wie Filteranzahl, Kernelgröße, Dropoutraten oder Aktivierungsfunktionen weitere optimierbare Parameter. Bei den rekurrenten Netzen gibt es zudem eine große Forschungsvielfalt an strukturellen Varianten des klassischen LSTMs [Goodfellow u. a., 2016, Kap.10]. Mithilfe kleinerer Stellschrauben wie Lernraten oder Batchgrößen kann ein KNN zudem praktisch ausoptimiert werden. Und auch die Repräsentationen besitzen Parameter wie Abtastraten, Fenstergrößen, Schrittweiten oder Anzahl an betrachteten Mels und MFC-Koeffizienten. Bei symbolischen Daten sind die Möglichkeiten von Varianten gar unerschöpflich.

Aber auch bei den Datensätzen gibt es Verbesserungsbedarf. Zum einen auf Seiten der Musikdaten: So bietet die Anwendung auf andere Datensätze weitere interessante Vergleichsmöglichkeiten. Beispielsweise wäre eine Erweiterung des Bach-Datensatzes um *BachBot*-Stücke [Liang u. a., 2017] oder Copes derzeit verlorenen *5000 Works in Bach Style*-Datensatz [Cope, 2012] spannend. Die Augmentierung der Daten auf diverse Art und Weisen ist dabei ebenso untersuchungswürdig [Goodfellow u. a., 2016, S.240ff]. Zum anderen ist aber auch die Gewinnung weiterer Geschmacksannotationen für weiterführende Arbeiten unumgänglich. Das Labeln von Geschmacksurteilen taugt bereits als Aufhänger einer eigenständigen Arbeit. Stellen sich weitere Ergebnisse ebenso als gewinnbringend heraus, könnte das Labeln von Geschmack unter Umständen sogar als standardmäßige Evaluationsroutine für kreative Musiksysteme vorgeschlagen werden. Von Vorteil wäre dabei auch die Aussage über das Gefallen von Segmenten eines Stückes

anstelle der ungenauen Angabe für immer ganze Stücke. Mit vielen Geschmacksdaten könnte zum Beispiel auch das Lernen verschiedener Geschmäcker verglichen werden. Vielleicht lernen sich gewisse Geschmäcker leichter. Die Gründe dafür wären es wiederum wert, durch tiefe Analyse beleuchtet zu werden.

Gerade an praktischen Analysetools und Methoden der Visualisierung mangelt es aber der Audiodomäne. So sollten auch hier grundlegend neue Techniken überdacht und ausgetestet werden. Vielleicht ist die hinderliche Abstraktion in Bilder überhaupt nicht in dem Maße gewinnbringend, wie gemeinhin angenommen wird. Ein interessanter Ansatz wäre es allemal, eine Analyse in akustische Signale zu kodieren. Aber auch die Verbesserung der bestehenden Tools vor allem auf Kompatibilitätsebene wäre für die effektive Forschung von großem Nutzen.

Mehr Erkenntnisse über die eigentlichen Funktionsprozesse, das heißt die Sichtbarmachung der verdeckten Schichten, könnte zu grundlegenden Erkenntnissen führen, die auch das ewige Problem der übergeordneten Zusammenhänge löst (vgl. Abschnitt 1.4). Um Metastrukturen abzubilden, kommt am ehesten die Verschaltung mehrerer Netze in Frage. So könnte auch die Ausgabe von verdeckten Schichten als Eingabe in neue Netze die nötige Komplexität weiter annähern. Dies deckt sich mit der zu Anfang des Abschnitts geforderten Interkompatibilität. So kann die Metapher vom Produzenten auch auf andere Bereiche wie zum Beispiel der Bilderzeugung, also *künstliche Maler*, aber auch auf *künstliche Schriftsteller*, *Bildhauer*, *Architekten* oder *Modeschöpfer* angewendet werden.

Bei dem naheliegenden Versuch den *künstlichen Musikproduzenten*, der auf *DeepBach* trainiert wurde, auf die *Computoser*-Erzeugnisse anzuwenden, zeigte sich kurioses: Er gab fast ausschließlich Werte nahe 1 aus. Ihm gefällt also alles, was der *Computoser* erzeugt, und das in außerordentlichem Maße. Dieser naive Fehlversuch ist Teil der Zukunftsvision eines *künstlichen Musikproduzenten*, der Aussagen des Gefallens und der subjektiven Qualität über die Grenzen verschiedener *künstlicher* (und/oder menschlicher) Komponisten trifft. Die Einschätzung neuer Stücke unbekannter Komponisten wäre demnach ein nächster Schritt in Richtung der Utopie der universellen KI.

A. Anhang

A.1. Bachness Korrelationsdiagramm

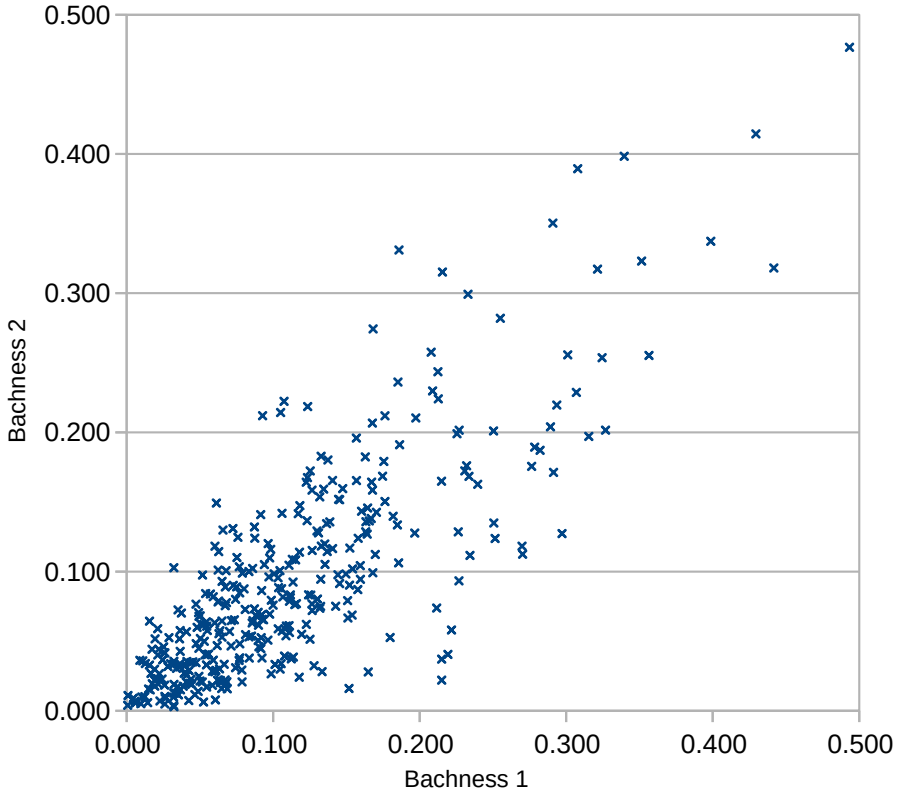


Abbildung A.1.: Korrelation der *Bachness* $\mathcal{B}$ für Pianoroll-MLP3 (1) und Pianoroll-MiniCNN (2).

A.2. Conv-Filter des MelSpec-MiniCNN

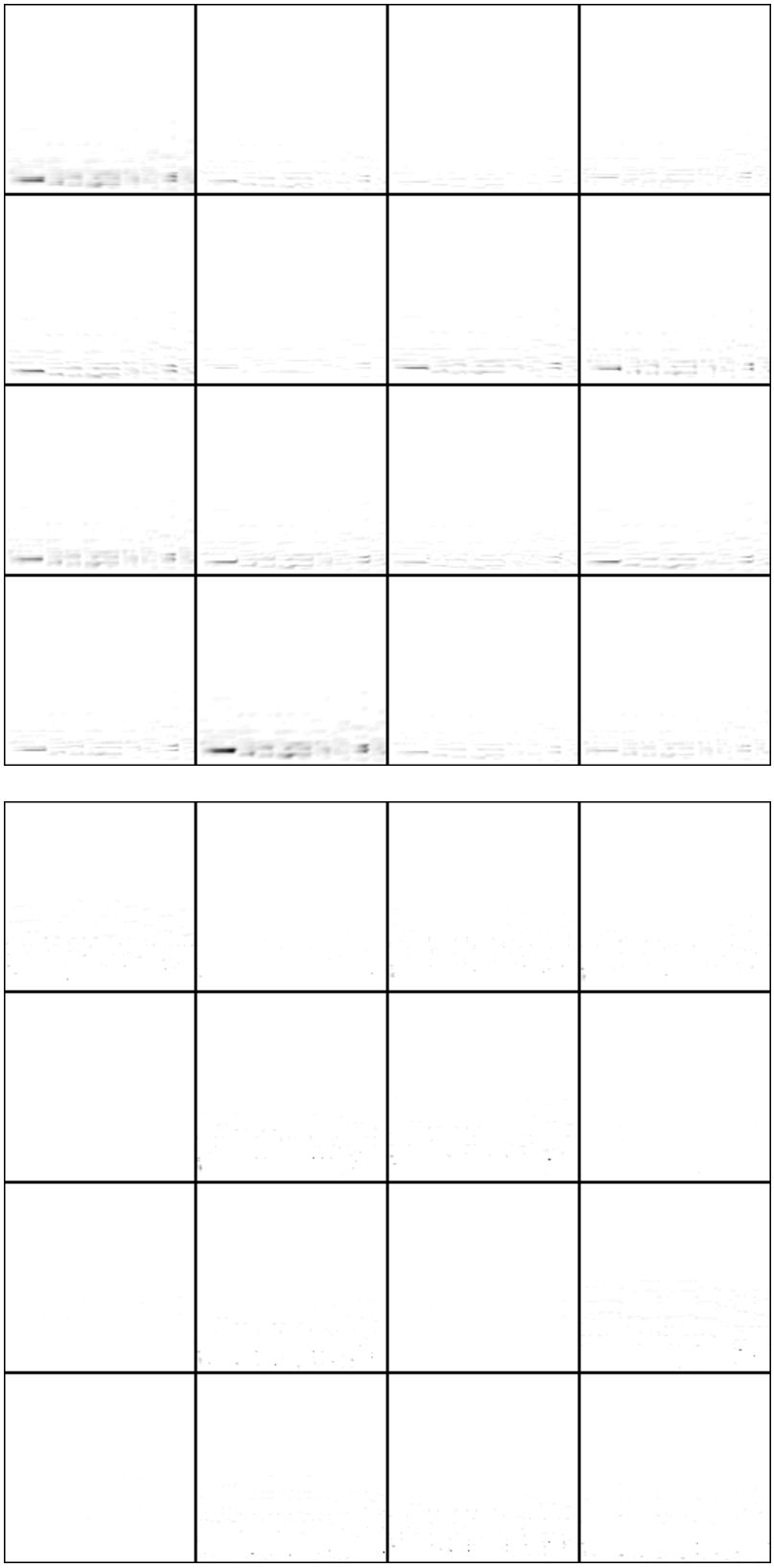


Abbildung A.2.: Aktivierungsanalyse der ersten (obere 16) und zweiten Conv-Schicht (untere 16) des MelSpec-MiniCNN. Höhere Aktivierungen sind dunkler.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AK	Algorithmische Komposition
BPM	Beats per Minute
BWV	Bach Werke Verzeichnis
CAM	Class Activation Mapping
CNN	Convolutional Neuronal Network
DAW	Digital Audio Workstation
DFT	Diskrete Fourier-Transformation
EA	Evolutionären Algorithmen
FT	Fourier-Transformation
FFT	Schnelle Fourier-Transformation
GAN	Generative Adversarial Networks
HMM	Hidden Markov Model
KI	Künstliche Intelligenz
KKR	Korrektklassifikationsrate
KNN	Künstliche neuronale Netze
KV	Köchelverzeichnis
LSTM	Long Short-Term Memory
MelS	Mel-Spektrogramm
MFCC	Mel-Frequenz-Cepstrum-Koeffizient
MIDI	Musical Instrument Digital Interface
MIR	Music Information Retrieval
MLP	Mehrlagige Perzeptron
NIFF	Notation Interchange File Format
PD-Roll	PianoDrumroll
P-Roll	Pianoroll
ReLU	Rectified Linear Unit
RNN	Rekurrentes Neuronales Netzwerk
SATB	Vierstimmiger Vokalsatz (Sopran, Alt, Tenor und Bass)
SMF	Standard-MIDI-Files
STFT	Kurzzeit-Fourier-Transformation
XML	eXtensible Markup Language

Literaturverzeichnis

[Abadi u. a. 2015] Abadi, Martín ; Agarwal, Ashish ; Barham, Paul ; Brevdo, Eugene ; Chen, Zhifeng ; Citro, Craig ; Corrado, Greg S. ; Davis, Andy u. a.: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. `https://www.tensorflow.org/` (zul. abgerufen am 9.6.2021). Version: 2015

[Allan u. Williams 2004] Allan, Moray ; Williams, Christopher K. I.: Harmonising Chorales by Probabilistic Inference. In: *Proceedings of the 17th International Conference on Neural Information Processing Systems*. Cambridge, MA, USA : MIT Press, 2004 (NIPS'04), S. 25–32

[Ames u. Domino 1992] In: Ames, Charles ; Domino, Michael: *Cybernetic Composer: An Overview*. Cambridge, USA : MIT Press, 1992, S. 186–205

[Ariza 2009] Ariza, Christopher: The Interrogator as Critic: The Turing Test and the Evaluation of Generative Music Systems. In: *Computer Music Journal* 33 (2009), Nr. 2, S. 48–70

[Armentano u. a. 2017] Armentano, Marcelo G. ; De Noni, Walter A. ; Cardoso, Hernán F.: Genre classification of symbolic pieces of music. In: *Journal of Intelligent Information Systems* 48 (2017), Nr. 6, S. 579–599

[Bach 1985] Bach, S. J. *389 Chorales: for SATB Voices; Choral Score*. Alfred Publishing Company, 1985 (Kalmus Classic Edition)

[Bäck u. a. 1997] Bäck, Thomas ; Fogel, David B. ; Michalewicz, Zbigniew: *Handbook of Evolutionary Computation*. Boca Raton, US : CRC Press, 1997

[Bahuleyan 2018] Bahuleyan, Hareesh: Music Genre Classification using Machine Learning Techniques. In: *CoRR* abs/1804.01149 (2018). `http://arxiv.org/abs/1804.01149` (zul. abgerufen am 9.6.2021)

[Berger 2001] In: Berger, Jonathan: *Who Cares if It Listens? An Essay on Creativity, Expectations, and Computational Modeling of Listening to Music*. Cambridge, USA : MIT Press, 2001

[Bogert u. a. 1963] Bogert, B.P. ; Healy, M.J. ; Tukey, J.W.: The quefrency alanysis of time series for echoes: Cepstrum, pseudo-autocovariance, cross-cepstrum and saphe cracking. In: *Proceedings of the symposium on time series analysis*. New York, USA : Wiley, 1963, S. 209–243

[Bozhanov 2014] Bozhanov, Bozhidar: Computoser – rule-based, probability-driven algorithmic music composition. In: *Computing Research Repository (CoRR)* abs/1412.3079 (2014)

[Butz 2015] Butz, Tilman: *Fourier Transformation for Pedestrians*. 2. Auflage. Basel, Schweiz : Springer International Publishing, 2015

[Cano u. a. 2007] Cano, José R. ; Herrera, Francisco ; Lozano, Manuel: Evolutionary stratified training set selection for extracting classification rules with trade off precision-interpretability. In: *Data & Knowledge Engineering* 60 (2007), Nr. 1, S. 90–108

[Chattopadhay u. a. 2018] Chattopadhay, Aditya ; Sarkar, Anirban ; Howlader, Prantik ; Balasubramanian, Vineeth N.: Grad-CAM++: Generalized Gradient-Based Visual Explanations for Deep Convolutional Networks. In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)* (2018)

[Chollet u. a. 2015] Chollet, François ; O'Malley, Tom ; Tan, Zhenyu ; Bileschi, Stan ; Gibson, Adam ; Allaire, JJ ; Rahman, Fariz ; Branchaud-Charron, Frederic ; Lee, Taehoon ; de Marmiesse, Gabriel u. a.: *Keras*. `https://keras.io` (zul. abgerufen am 9.6.2021), 2015

[Collins 2010] Collins, Nick: *Introduction to Computer Music*. Chichester, UK : John Wiley & Sons Ltd, 2010

[Collins 2018] Collins, Nick: „...there is no reason why it should ever stop": Large-scale algorithmic composition. In: *Journal of Creative Music Systems* 3 (2018), Nr. 1

[Cope 1996] Cope, David: *Experiments in Musical Intelligence*. Middleton, WI, USA : A-R Editions, 1996

[Cope 2001] Cope, David: *Virtual Music – Computer Synthesis of Musical Style*. Cambridge, USA : MIT Press, 2001

[Cope 2012] Cope, David: *5000 Works in Bach Style*. `http://artsites.ucsc.edu/faculty/cope/5000.html` (zul. abgerufen am 9.6.2021). Version: 2012. – Der Link führt zur Projektwebsite. Der eigentliche Datensatz ist dort derzeit nicht abrufbar.

[Cuthbert u. Ariza 2010] Cuthbert, Michael S. ; Ariza, Christopher: music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data. In: Downie, J. S. (Hrsg.) ; Veltkamp, Remco C. (Hrsg.): *11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, 2010, S. 637–642

[Darwin 1859] Darwin, Charles R.: *On the Origin of Species*. London, UK : John Murray, 1859

[Diaz-Jerez 2011] Diaz-Jerez, Gustavo: Composing with Melomics: Delving into the Computational World for Musical Inspiration. In: *Leonardo Music Journal* 21 (2011), S. 13–14

[Dieleman u. Schrauwen 2014] Dieleman, Sander ; Schrauwen, Benjamin: End-to-end learning for music audio. In: *ICASSP* IEEE, 2014, S. 6964–6968

[Donahue u. a. 2019] Donahue, Chris ; McAuley, Julian ; Puckette, Miller: Adversarial Audio Synthesis. In: *ICLR*, 2019

[Dong u. a. 2018] Dong, Hao-Wen ; Hsiao, Wen-Yi ; Yang, Li-Chia ; Yang, Yi-Hsuan: MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. In: *roceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018

[Ebcioğlu 1988] Ebcioğlu, Kemal: An Expert System for Harmonizing Four-Part Chorales. In: *Computer Music Journal* 12 (1988), Nr. 3, S. 43–51

[Ebcioğlu 1990] Ebcioğlu, Kemal: An expert system for harmonizing chorales in the style of J.S. Bach. In: *The Journal of Logic Programming (Special Issue: Logic Programming Applications)* 8 (1990), Nr. 1, S. 145–185

[Eck u. Schmidhuber 2002] Eck, Douglas ; Schmidhuber, Juergen: Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. In: *IEEE International Workshop on Neural Networks for Signal Processing*. Martigny, Valais, Switzerland, 2002

[Eggebrecht 1967] Eggebrecht, Hans H. (Hrsg.): *Riemann Musiklexikon – Sachteil*. Mainz : Schott, 1967

[Exner 1894] Exner, Siegmund: *Entwurf zu einer physiologischen Erklärung der psychischen Erscheinungen*. Leipzig : F. Deuticke, 1894

[Fernandez u. Vico 2013] Fernandez, J.D. ; Vico, F.: AI Methods in Algorithmic Composition: A Comprehensive Survey. In: *Journal of Artificial Intelligence Research* 48 (2013), Nov, S. 513–582

[Gannon 1990] Gannon, Peter: *Band-in-a-Box*. Hamilton, Canada : PG Music, Inc., 1990

[Gebesmair 2001] Gebesmair, Andreas: *Grundzüge einer Soziologie des Musikgeschmacks*. Wiesbaden : Springer Fachmedien, 2001

[Geman u. Geman 1984] Geman, Stuart ; Geman, Donald: Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6 (1984), S. 721–741

[Goodfellow u. a. 2016] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016 `http://www.deeplearningbook.org` (zul. abgerufen am 9.6.2021)

[Goodfellow u. a. 2014] Goodfellow, Ian ; Pouget-Abadie, Jean ; Mirza, Mehdi ; Xu, Bing ; Warde-Farley, David ; Ozair, Sherjil ; Courville, Aaron ; Bengio, Yoshua: Generative Adversarial Nets. In: Ghahramani, Z. (Hrsg.) ; Welling, M. (Hrsg.) ; Cortes, C. (Hrsg.) ; Lawrence, N. (Hrsg.) ; Weinberger, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 27, Curran Associates, Inc., 2014

[Hadjeres u. a. 2017] Hadjeres, Gaëtan ; Pachet, François ; Nielsen, Frank: DeepBach: a Steerable Model for Bach Chorales Generation. In: Precup, Doina (Hrsg.) ; Teh, Yee W. (Hrsg.): *Proceedings of the 34th International Conference on Learning* Bd. 70, PMLR, 2017 (Proceedings of Machine Learning Research), S. 1362–1371

[Han u. a. 2017] Han, Yoonchang ; Kim, Jaehun ; Lee, Kyogu: Deep Convolutional Neural Networks for Predominant Instrument Recognition in Polyphonic Music. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25 (2017), Nr. 1, S. 208–221

[Harris 1978] Harris, Fredric J.: On the Use of Windows for Harmonic Analysis With the Discrete Fourier Transform. In: *Proc. IEEE*, 1978, S. 51–83

[Hazard u. Kimport 1999] Hazard, Chris ; Kimport, Catherine: *Fractal Music*. `http://www.tursiops.cc/fm/` (zul. abgerufen am 9.6.2021). Version: 1999. – Derzeit abrufbar über `http://web.archive.org/web/20160304043456/http://www.tursiops.cc/fm/`

[Helmholtz 1865] Helmholtz, Hermann v.: *Die Lehre von den Tonempfindungen als physiologische Grundlage für die Theorie der Musik*. Braunschweig : F. Vieweg, 1865

[Herremans u. a. 2014] Herremans, Dorien ; Martens, David ; Sörensen, Kenneth: Dance Hit Song Prediction. In: *Journal of New Music Research* 43 (2014), Nr. 3, S. 291–302

[Hild u. a. 1992] Hild, H. ; Feulner, J. ; Menzel, D.: HARMONET: a neural net for harmonizing chorals in the style of J.S.Bach. In: *Advances in Neural Information Processing* 4 (1992)

[Hiller 1996] Hiller, Paul: *Arvo Pärt (Oxford Studies of Composers)*. Oxford Univeristy Press, 1996

[Hochreiter u. Schmidhuber 1997] Hochreiter, Sepp ; Schmidhuber, Jürgen: Long Short-Term Memory. In: *Neural Computation* 9 (1997), Nr. 8, S. 1735–1780

[Hofstadter 2001] In: Hofstadter, Douglas: *A Few Standard Questions and Answers*. Cambridge, USA : MIT Press, 2001

[Hutmacher 2019] Hutmacher, Fabian: Why Is There So Much More Research on Vision Than on Any Other Sensory Modality? In: *Front. Psychol.* (2019)

[Iizuka u. a. 2017] Iizuka, Satoshi ; Simo-Serra, Edgar ; Ishikawa, Hiroshi: Globally and Locally Consistent Image Completion. In: *ACM Transactions on Graphics* 36 (2017), Nr. 4

[Jacob 1996] Jacob, Bruce L.: Algorithmic Coposition as a Model Of Creativity. In: *Organised Sound: Special Issue on Algorithmic Composition* 1 (1996), Nr. 3

[Jannach u. a. 2010] Jannach, Dietmar ; Zanker, Markus ; Felfernig, Alexander ; Friedrich, Gerhard: *Recommender Systems: An Introduction*. Cambridge University Press, 2010

[Kandel u. Schwartz 1995] Kandel, Eric R. ; Schwartz, James H. ; Jessel, Thomas M. (Hrsg.): *Neurowissenschaften. Eine Einführung*. Heidelberg, Berlin, Oxford : Spektrum Akademischer Verlag, 1995

[Karras u. a. 2018] Karras, Tero ; Laine, Samuli ; Aila, Timo: A Style-Based Generator Architecture for Generative Adversarial Networks. In: *CoRR* abs/1812.04948 (2018). `http://arxiv.org/abs/1812.04948` (zul. abgerufen am 9.6.2021)

[Kindermans u. a. 2019] Kindermans, Pieter-Jan ; Hooker, Sara ; Adebayo, Julius ; Alber, Maximilian ; Schütt, Kristof T. ; Dähne, Sven ; Erhan, Dumitru ; Kim, Been: The (un) reliability of saliency methods. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 2019, S. 267–280

[Kingma u. Ba 2015] Kingma, Diederik P. ; Ba, Jimmy: Adam: A Method for Stochastic Optimization. In: *3rd International Conference for Learning Representations*. San Diego, USA, 2015

[Klatt 1977] Klatt, Dennis H.: Review of the ARPA speech understanding project. In: *The Journal of the Acoustical Society of America* 62 (1977), Nr. 6, S. 1345–1366

[Kordos 2016] Kordos, Miroslaw: Data Selection for Neural Networks. In: *Schedae Informaticae* 25 (2016), S. 153–164

[Krizhevsky u. a. 2012] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F. (Hrsg.) ; Burges, C. J. C. (Hrsg.) ; Bottou, L. (Hrsg.) ; Weinberger, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 25, Curran Associates, Inc., 2012

[KV 1862] KV, Köchelverzeichnis,: *Chronologisch-thematisches Verzeichnis sämtlicher Tonwerke Wolfgang Amade Mozarts. Nebst Angabe der verloren gegangenen, angefangenen, übertragenen, zweifelhaften und unterschobenen Compositionen desselben.* Wiesbaden : Breitkopf & Härtel, 1862/1965

[Kühn 1987] Kühn, Clemens: *Formenlehre der Musik*. Kassel : Bärenreihter, 1987

[Küpfmüller u. Kohn 1932] Küpfmüller, Karl ; Kohn, Gerhard: *Theoretische Elektrotechnik und Elektronik – Eine Einführung*. 15. Auflage. Berlin, Heidelberg : Springer-Verlag, 1932–2000

[Laplante 2014] Laplante, Audrey: Improving Music Recommender Systems: What Can We Learn from Research on Music Tastes? In: *15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, 2014

[Lapuschkin u. a. 2016] Lapuschkin, Sebastian ; Binder, Alexander ; Montavon, Grégoire ; Müller, Klaus-Robert ; Samek, Wojciech: The LRP Toolbox for Artificial Neural Networks. In: *Journal of Machine Learning Research* 17 (2016)

[le Cun 1989] le Cun, Yan: Generalization and network design strategies / Connectionist Research Group. University of Toronto, 1989 (CRG-TR-89-4). – Forschungsbericht

[Lee u. a. 2009] Lee, Honglak ; Pham, Peter ; Largman, Yan ; Ng, Andrew Y.: Unsupervised feature learning for audio classification using convolutional deep belief networks. In: *22nd International Conference on Neural Information Processing Systems*, 2009, S. 1096–1104

[Lewis 2000] Lewis, George E.: Too Many Notes: Computer, Complexity and Culture in Voyager. In: *Leonardo Music Journal* 10 (2000), 33–39. `http://muse.jhu.edu/article/20320/pdf` (zul. abgerufen am 9.6.2021)

[Lewis 1988] Lewis, J. P.: Creation by refinement: A creativity paradigm for gradient descent learning networks. In: *IEEE ICNN* Bd. 2, 1988, S. 229–233

[Lewis 1991] In: Lewis, J P.: *Creation by Refinement and the Problem of Algorithmic Music Composition.* Cambridge, USA : MIT Press, 1991

[Li u. a. 2010] Li, Tom L. ; Chan, Antoni B. ; Chun, Andy H.: Automatic Musical Pattern Feature Extraction Using Convolutional Neural Network. In: *Proc. International MultiConference of Engineers and Computer Scientists (IMECS)* Bd. 1. Hong Kong, 2010

[Li u. Wu 2015] Li, Xiangang ; Wu, Xihong: Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* IEEE, 2015, S. 4520–4524

[Liang u. a. 2017] Liang, Feynman ; Gotham, Mark ; Johnson, Matthew ; Shotton, Jamie: Automatic Stylistic Composition of Bach Chorales with Deep LSTM. In: *18th International Society for Music Information Retrieval Conference*, 2017

[Logan 2000] Logan, Beth: Mel Frequency Cepstral Coefficients for Music Modeling. In: *Proc. 1st Int. Symposium Music Information Retrieval* (2000)

[Loughran u. O'Neill 2017] Loughran, Róisín ; O'Neill, Michael: Limitations from Assumptions in Generative Music Evaluation. In: *Journal of Creative Music Systems* 2 (2017)

[Loughran u. O'Neill 2020] Loughran, Róisín ; O'Neill, Michael: Evolutionary music: applying evolutionary computation to the art of creating music. In: *Genetic Programming and Evolvable Machines* (2020), S. 55–85

[MacCallum u. a. 2012] MacCallum, Robert M. ; Mauch, Matthias ; Burt, Austin ; Leroi, Armand M.: Evolution of music by public choice. In: *Proceedings of the National Academy of Sciences* 109 (2012), Nr. 30, S. 12081–12086

[Markov 1913] Markov, Andrei A.: Classical Text in Translation: An Example of Statistical Investigation of the Text *Eugene Onegin* Concerning the Connection of Samples in Chains. In: *Science in Context* 19 (1913/2006), Nr. 4, S. 591–600

[Marolt u. a. 2002] Marolt, Matija ; Kavcic, Alenka ; Privosnik, Marko: *Neural networks for note onset detection in piano music.* `https://www.researchgate.net/profile/Matija_Marolt/publication/2473938_Neural_Networks_for_Note_Onset_Detection_in_Piano_Music/links/00b49525efccc79fed000000.pdf` (zul. abgerufen am 9.6.2021). Version: 2002

[Maurer 1999] Maurer, John A.: *A Brief History of Algorithmic Composition.* `https://ccrma.stanford.edu/~blackrse/algorithm.html` (zul. abgerufen am 9.6.2021). Version: 1999/2004

[McFee u. a. 2020] McFee, Brian ; Lostanlen, Vincent ; Metsai, Alexandros ; McVicar, Matt ; Balke, Stefan ; Thomé, Carl ; Raffel, Colin ; Zalkow, Frank ; Malek, Ayoub u. a.: *librosa/librosa: 0.8.0.* `https://doi.org/10.5281/zenodo.3955228` (zul. abgerufen am 9.6.2021). Version: 2020

[McKay 2010] McKay, Cory: *Automatic music classification with jMIR*. Kanada, McGill University, Diss., 2010

[Middlebrook u. Sheik 2019] Middlebrook, Kai ; Sheik, Kian: Song Hit Prediction: Predicting Billboard Hits Using Spotify Data. In: *CoRR* abs/1908.08609 (2019). `http://arxiv.org/abs/1908.08609` (zul. abgerufen am 9.6.2021)

[Miranda u. Biles 2007] Miranda, Eduardo R. (Hrsg.) ; Biles, John A. (Hrsg.): *Evolutionary Computer Music*. London, UK : Springer, 2007

[MMA 1991] MMA, MIDI Manufacturers Association: *General MIDI 1*. 1991 `https://www.midi.org/specifications/midi1-specifications/general-midi-specifications/general-midi-1` (zul. abgerufen am 9.6.2021)

[Niemann 1983] Niemann, Heinrich: *Klassifikation von Mustern*. Berlin, Heidelberg : Springer-Verlag, 1983

[Nishijima u. Watanabe] Nishijima, Maskao ; Watanabe, Kazuyuki: Interactive music composer based on neural net- works. In: *Fujitsu Scientific Technical Journal* 29, Nr. 2, S. 189–192

[van den Oord u. a. 2016] Oord, Aäron van den ; Dieleman, Sander ; Zen, Heiga ; Simonyan, Karen ; Vinyals, Oriol ; Graves, Alex ; Kalchbrenner, Nal ; Senior, Andrew W. ; Kavukcuoglu, Koray: WaveNet: A Generative Model for Raw Audio. In: *CoRR* abs/1609.03499 (2016). `http://arxiv.org/abs/1609.03499` (zul. abgerufen am 9.6.2021)

[O'Shaughnessy 1987] O'Shaughnessy, D.: *Speech Communication: Human and Machine*. Addison-Wesley Publishing Company, 1987 (Addison-Wesley series in electrical engineering)

[Ostermann u. a. 2017] Ostermann, Fabian ; Vatolkin, Igor ; Rudolph, Günter: Evaluation Rules for Evolutionary Generation of Drum Patterns in Jazz Solos. In: Correia, Joao (Hrsg.) ; Ciesielski, Vic (Hrsg.) ; Liapis, Antonios (Hrsg.): *Computational Intelligence in Music, Sound, Art and Design. EvoMUSART 2017*. Cham, Germany : Springer, 2017, S. 246–261

[Patel u. Demorest 2013] In: Patel, Aniruddh D. ; Demorest, Steven M.: *Comparative Music Cognition: Cross-Species and Cross-Cultural Studies*. Elsevier Inc, 2013, S. 647–681

[Pons 2018] Pons, Jordi: Neural Networks For Music: A Journey Through Its History. In: *Medium: towards data science* (2018). `https://towardsdatascience.com/neural-networks-for-music-a-journey-through-its-history-91f93c3459fb` (zul. abgerufen am 9.6.2021)

[Prechelt 1996] *Kapitel* Early Stopping—But When? In: Prechelt, Lutz: *Neural Networks: Tricks of the Trade*. Bd. LNCS 1524. Berlin, Heidelberg : Springer, 1996, S. 55–69

[Prokofiev 1978] Prokofiev, Sergei ; Blok, V. (Hrsg.): *Sergei Prokofiev: materials, articles, interviews*. Moskau, Russland : Progress Publishers, 1978

[Rabiner u. Juang 1986] Rabiner, Lawrence R. ; Juang, Biing-Hwang: An Introduction to Hidden Markov Models. In: *IEEE ASSP Magazine* 3 (1986), Nr. 1

[Raffel 2016] Raffel, Colin: *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*, Columbia University, Diss., 2016

[Richard O. Duda 2001] Richard O. Duda, David G. S. Peter E. Hart H. Peter E. Hart: *Pattern Classification.* New York, USA : John Wiley & Sons, 2001

[Roberts u. a. 2018] Roberts, Adam ; Engel, Jesse ; Raffel, Colin ; Hawthorne, Curtis ; Eck, Douglas: A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music. In: *Proc. 35th International Conference on Machine Learning.* Stockholm, Schweden, 2018

[Rowe 2001] Rowe, Robert: *Machine Musicianship.* Cambridge, USA : MIT Press, 2001

[Schillinger 1948] Schillinger, Joseph: *The mathematical basis of the arts.* New York, USA : Philosophical Library, 1948

[Schönberg 1992] Schönberg, Arnold ; Vojtech, Ivan (Hrsg.): *Stil und Gedanke.* Fischer Verlag, 1992

[Schürger 2016] Schürger, T.: *SoundHelix – Algorithmic random music composer – Documentation*, 2016. `https://www.soundhelix.com/doc/running` (zul. abgerufen am 9.6.2021)

[Selvaraju u. a. 2019] Selvaraju, Ramprasaath R. ; Cogswell, Michael ; Das, Abhishek ; Vedantam, Ramakrishna ; Parikh, Devi ; Batra, Dhruv: Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In: *International Journal of Computer Vision* 128 (2019), Nr. 2, S. 336–359

[Shannon 1949] Shannon, Claude E.: Communication in the Presence of Noise. In: *Proc. IRE* Bd. 37, 1949

[Shibata 1991] Shibata, Naoki: A neural network-based method for chord/note scale association with melodies. In: *NEC Research and Development* 32 (1991), Nr. 3, S. 453–459

[Simonyan u. a. 2014] Simonyan, Karen ; Vedaldi, Andrea ; Zisserman, Andrew: Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. In: *CoRR* abs/1312.6034 (2014). `https://arxiv.org/abs/1312.6034` (zul. abgerufen am 9.6.2021)

[Simonyan u. Zisserman 2015] Simonyan, Karen ; Zisserman, Andrew: Very Deep Convolutional Networks For Large-Scale Image Recognition. In: *Proc. ICLR 2015* (2015)

[Smilkov u. a. 2017] Smilkov, Daniel ; Thorat, Nikhil ; Kim, Been ; Viégas, Fernanda B. ; Wattenberg, Martin: SmoothGrad: removing noise by adding noise. In: *CoRR* abs/1706.03825 (2017). `http://arxiv.org/abs/1706.03825` (zul. abgerufen am 9.6.2021)

[Srivastava u. a. 2014] Srivastava, Nitish ; Hinton, Geoffrey ; Krizhevsky, Alex ; Sutskever, Ilya ; Salakhutdinov, Ruslan: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In: *Journal of Machine Learning Research* (2014), S. 1929–1958

[Stevens u. a. 1937] Stevens, S. S. ; Volkmann, J. ; Newman, E. B.: A Scale for the Measurement of the Psychological Magnitude Pitch. In: *The Journal of the Acoustical Society of America* (1937)

[Stift. Warentest 2019] Stift. Warentest: Neugierige Helfer. In: *Stiftung Warentest* 04/2019 (2019)

[Stumpf 1883] Stumpf, Carl: *Tonpsychologie, Band 1 und 2.* Leipzig : S. Hirzel, 1883/1890

[Sweet 2014] Sweet, Michael: *Writing Interactive Music for Video Games: A Composer's Guide (Game Design).* Boston, USA : Addison-Wesley Professional, 2014

[Taube 2013] Taube, Heinrich: *Notes from the Metalevel: An Introduction to Computer Composition.* Taylor & Francis, 2013. – zuerst veröffentlicht in 2005

[Tipei 1975] Tipei, Sever: MP1: A computer program for music composition. In: *Proceedings of the Annual Music Computation Conference*, 1975

[Todd 1988] Todd, Peter M.: A sequential network design for musical applications. In: *Connectionist Models Summer School*, 1988, S. 76–84

[Vinyals u. a. 2015] Vinyals, Oriol ; Toshev, Alexander ; Bengio, Samy ; Erhan, Dumitru: Show and tell: A neural image caption generator. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, S. 3156–3164

[Wang 2003] Wang, Avery Li-Chun: An Industrial-Strength Audio Search Algorithm. In: *Proc. International Symposium on Music Information Retrieval (ISMIR)*, 2003

[Wang u. a. 2020] Wang, Haofan ; Wang, Zifan ; Du, Mengnan ; Yang, Fan ; Zhang, Zijian ; Ding, Sirui ; Mardziel, Piotr ; Hu, Xia: Score-CAM: Score-Weighted Visual Explanations for Convolutional Neural Networks. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, S. 111–119

[Wang u. a. 2017] Wang, Suhang ; Wang, Yilin ; Tang, Jiliang ; Shu, Kai ; Ranganath, Suhas ; Liu, Huan: What Your Images Reveal: Exploiting Visual Contents for Point-of-Interest Recommendation. In: *Proceedings of the 26th International Conference on World Wide Web.* Republic and Canton of Geneva, CHE : International World Wide Web Conferences Steering Committee, 2017, S. 391–400

[Wang 2011] Wang, Wenwu: *Machine Audition: Principles, Algorithms and Systems.* Hershey, PA, USA : IGI Global, 2011

[Wikimedia 2008] Wikimedia, User: Chabacano: *diagram showing overfitting of a classifier.* `https://commons.wikimedia.org/wiki/File:Overfitting.svg` (zul. abgerufen am 9.6.2021). Version: 2008

[Wikimedia 2005] Wikimedia, User: Chrislb: *Schema eines künstlichen Neurons: Künstliches Neuron mit Index j.* `https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_deutsch.png` (zul. abgerufen am 9.6.2021). Version: 2005

[Wikimedia 2007] Wikimedia, User: Gringer: *SVG version of Overfitting.png*. `https://commons.wikimedia.org/wiki/File:Overfitting_svg.svg` (zul. abgerufen am 9.6.2021). Version: 2007

[Wolf 1919] Wolf, Johannes: *Handbuch der Notationskunde*. Bd. 2. Leipzig : Breitkopf & Härtel, 1919

[Wolfram 2002] Wolfram, Stephen: *A New Kind of Science*. Wolfram Media, 2002 `http://tones.wolfram.com/about/how-it-works` (zul. abgerufen am 9.6.2021)

[Xenakis 1971] Xenakis, Iannis: *Formalized Music: Thought and Mathematics in Composition*. Bloomington, London, UK : Indiana Univeristy Press, 1971

[Xu u. a. 2020] Xu, Han ; Ma, Yao ; Liu, Hao-Chen ; Deb, Debayan ; Liu, Hui ; Tang, Ji-Liang ; Jain, Anil K.: Adversarial Attacks and Defenses in Images, Graphs and Text: A Review. In: *International Journal of Automation and Computing* 17 (2020), S. 151–178

[Ye 2020] Ye, Andre: *MachineCurve, created with keras viz. In: Every ML Engineer Needs to Know Neural Network Interpretability*. `https://medium.com/analytics-vidhya/every-ml-engineer-needs-to-know-neural-network-interpretability-afea2ac0824e` (zul. abgerufen am 9.6.2021). Version: 2020

[Zhou u. a. 2016] Zhou, B. ; Khosla, A. ; A., Lapedriza. ; Oliva, A. ; Torralba, A.: Learning Deep Features for Discriminative Localization. In: *Conference on Computer Vision and Pattern Recognition* (2016)

[Zicarelli 1987] Zicarelli, David: M and Jam Factory. In: *Computer Music Journal* 11 (1987), Nr. 4, S. 13–29

Eidesstattliche Versicherung
(Affidavit)

Ostermann, Fabian

Name, Vorname
(Last name, first name)

156452

Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present Bachelor's/Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der ~~Bachelor-~~/Masterarbeit*:
(Title of the Bachelor's/ Master's* thesis):

Modellierung des menschlichen Musikgeschmacks mit neuronalen Netzen

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

Dortmund, 9.6.2021

Ort, Datum
(Place, date)

F. Ost

Unterschrift
(Signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

Dortmund, 9.6.2021

Ort, Datum
(Place, date)

F. Ost

Unterschrift
(Signature)

**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.