Universität Hamburg

Bachelorprojekt

Big Data

Klassifizierung von Musik-Genres mit ANN

Abgabe: 13. Mai 2018

Fachbereich Informatik

Authoren:
6818432, Singh Jagmit
6824327, Fuhrmann Gian-Luca

Aufsicht:
Dr. Julian Kunkel

Inhaltsverzeichnis

1 Einleitung

Zu Beginn des Projektes standen wir zunächst vor der Aufgabe ein Thema zu finden, mit welchem wir uns für das kommende halbe Jahr befassen würden. Unter dem breiten Themengebiet "Big Data" entschlossen wir uns, über die Wahl einer interessanten Datendomain, potentiellen Themen anzunähern. Faszinierender als textuelle, rein numerische, oder Bilddaten, erschien uns der Bereich der Audio–Daten — spezieller Musik. Wie für viele Menschen, ist Musik ein großer und wichtiger Teil unserer beider Leben, und heute mehr als je zuvor sind die Möglichkeiten Musik zu beziehen mannigfaltig. Ein weiterer Aspekt, der diese Domain besonders interessant macht, ist die komplexe Beziehung zwischen den physikalischen Phänomenen der Akustik, menschlicher auditiver Wahrnehmung und informationstechnischer Codierung der Signale. Wir wussten also, dass — so wenn wir uns für Musik als "Data" entschieden — *erstes Ziel* unseres Projektes sein würde, diese Beziehungen näher zu untersuchen und uns so eine Grundlage zur Bearbeitung des letztlich gewählten praktischen Themas zu schaffen.

Die Definition von "Big Data" gebietet es, dass wir ausreichend große Datenmengen zur Verarbeitung ansammeln und somit zum *zweiten Ziel* unseres Projektes gelangen: Dem Aufbau eines Datensatzes. Den kommerziellen Interessen hinter Musik geschuldet, ist es auf Grund von Urheberrechtsgesetzen, trotz der weiten Verfügbarkeit, schwierig, an nutzbare Daten zu kommen. Die Beschaffung stellt also, zusätzlich zur Vor–Verarbeitung und Nutzbarmachung, eine Schwierigkeit da, die es zu überkommen heißt.

Nach Rücksprache mit den Betreuern des Moduls entschieden wir uns, aus mehreren möglichen Implementationszielen, für die Umsetzung eines Musik-Genre-Klassifizierers unter Verwendung künstlicher neuronaler Netze. Hierfür müssen Grundlagen und Architekturvarianten erkundet und im Kontext unseres Implementationszieles abgewogen werden.

- Untersuchung von Grundlagen
  - der Akustik
  - der menschlichen auditiven Wahrnehmung
  - digitaler Audio–Signal–Codierung
- Beziehen, Verarbeiten und Nutzbarmachen von Musik
- Implementieren eines neuronalen Klassifizierers für Musik–Genres

Diese drei Ziele sollen im Laufe dieses Projektes so gut möglich erreicht werden.

2 Grundlagen

Im Sinne des ersten Zieles unseres Projekts, haben wir uns zunächst einen Überblick über die zugrundeliegenden Phänomene des Hörens und digitaler Musik verschafft. Zusätzlich zu den in der Einleitung beschriebenen Themen, werden in diesem Abschnitt des Berichts auch im Groben die Funktionsweise von artifiziellen Neuronalen Netzwerken untersucht, welche zur Klassifizierung der gesammelten Daten – und auf Basis unseres erarbeiteten Verständnisses – genutzt werden sollten.

2.1 Relevante Grundlagen der Akustik

Um die Funktionsweise des menschlichen Ohrs zu verstehen, muss zunächst das physikalischen Phänomen betrachtet werden, dessen Informationen das auditive System verarbeitet – Schall[(1)]. Wir beschränken uns hier auf eine kleine Auswahl an Gesetzmäßigkeiten, welche für das Verständnis in diesem Kontext genügen.

Schall ist eine Schwingung der Teilchendichte im tragenden Medium, ausgelöst von Vibration eines Objektes, welches diese Vibration durch sich fortpflanzende Verdrängung der Teilchen als Longitudinal-Welle im Medium überträgt. [(2)] [(3)] Als solche ist die Übertragung offensichtlich abhängig von den Eigenschaften des Trägermediums. [1] Wichtigste Kenngröße des Schalls ist der *Schallwechseldruck* $\widetilde{p} = \frac{F}{A}$ $[Pa = \frac{N}{m^2}]$, als periodische Abweichung von der auf eine Fläche A $[m^2]$ wirkende Kraft F $[N]$, im Vergleich zum statischen Druck des Mediums p_0. Orientiert an der Hörschwelle des Menschen ($\widetilde{p}_0 = 2 * 10^{-5}\ Pa$), wird der *Schalldruckpegel* als logarithmisches Energieverhältnis angegeben: $L_p = 10 * lg(\frac{\widetilde{p}^2}{\widetilde{p}_0^2})$ $[dB]$. [3, S.4]

Der Schalldruck (*Amplitude* der Welle) entspricht dabei in der menschlichen Empfindung der *Lautstärke*. Die *Freuquenz* $[Hz]$ der ausgelösten Welle entspricht dem grundsätzlichen *Ton* des Geräusches. [4, S.263] [(4)] Wichtig ist hierbei, dass in der Regel nicht eine reine Welle mit klarer Frequenz vorliegt, sondern eine Überlagerung mehrerer Wellen. Die Amplituden addieren sich in dem Fall.

Die *Schallkennimpedanz* $Z = \frac{p}{\vec{v}} = \rho_0 * c$ $[\frac{kg}{m^2 s}]$[(5)] beschreibt den vom Ausbreitungs-

[(1)] "[...] acoustics is a study of sound [Schall] generation and propagation in different media" [1, S.34]

[(2)] "**2.01 sound.** (a) Oscillation in pressure, stress, particle displacement, particle velocity, etc., propagated in a medium with internal forces (e.g., elastic or viscous), or the superposition of such propagated oscillation." [2]

[(3)] Vgl Abb.6,S.31

[(4)] Weitere komplexere klangbeschreibende Eigenschaften sind in 2.2.3 angerissen.

[(5)] ρ: Dichte, c: Schallgeschwindigkeit; für planare Wellen [5, 5.10, S.126]

medium abhängigen Widerstand, der bei der Schallausbreitung überwunden werden muss. $\vec{v} = \frac{\partial \delta}{\partial t}$ ist dabei die *Schallschnelle* (die Geschwindigkeit der Teilchen des Mediums), mit δ als Auslenkung der Teilchen im Medium [6]. [3, S.6]

Mit Hilfe der *Schallkennimpendanz* zweier Medien[6], können wir den *Reflexionsfaktor* $r = \frac{Z_2 - Z_1}{Z_2 + Z_1}$ an der Grenzfläche berechnen. $r = 1$ entspricht dabei der totalen Reflexion an der Grenzfläche, und $r = 0$ bedeuted, dass die Welle sich vollständig im zweiten Medium fortpflanzt (keine Reflexion). [3, S.26] Es ist zu erkennen, dass für $Z_2 \gg Z_1 : r \to 1$ gilt.

2.2 Auditve Wahrnehmung des Menschen

Mit den Grundlagen aus 2.1, kann nun der Aufbau des menschlichen Ohrs betrachtet und die Aufgaben der einzelnen Teile untersucht werden. Auch soll die Verarbeitung der in Potentialen codierten Schallinformationen im Gehirn, zumindest grundlegend und strukturell, beschrieben werden. Abschließend werden noch weitere interessante Erkenntnisse aus der Psychoakustik vorgestellt.

2.2.1 Außen-, Mittel-, Innenohr

Aufgabe des Ohres[7] ist es in erster Linie, den variierenden Schalldruck auf neuronale Impulse abzubilden (*Transduktion*). Die eigentliche Wandlung in elektrische Signale erfolgt dabei allerdings erst im Innenohr. Zunächst wird der Schall über die Ohrmuschel[8] und den etwa 4cm langen *äußeren Gehörgang* tiefer in den Schädel geleitet. Die Haut des Gangs, welche an dessen Ende direkt in das *Trommelfell* übergeht, schmiegt sich im Verlauf nach innen immer enger am Knochen an und absorbiert so kaum Schallenergie. Die drei *Gehörknöchelchen – Hammer*, *Ambos* und *Steigbübel* – liegen direkt hinter dem Trommelfell, welches den Übergang vom Außen- zum **Mittelohr** darstellt. Die weniger als 0.1mm dicke Membran wird durch den eintreffenden Schall in Schwingung versetzt, und überträgt diese an den mit ihr verbundenen Hammer. Aufgabe des Mittelohrs ist es, über die Mechanik der Gehörknöchelchen eine *Impedanzanpassung* durchzuführen. Das letzte der verbundenen Knöchelchen – der Steigbügel – sitzt auf einer weiteren Membran. Dieses *ovale Fenster* bildet den Übergang zu der mit Flüssigkeit gefüllten *Gehörschnecke*. [8]

Wie in 2.1 beschrieben, kann es beim Übergang zweier Trägermedien zu Reflexion

[6] Von Medium 1 mit Z_1 zu Medium 2 mit Z_2
[7] Vgl. im folgenden auch Abb.7, S.31
[8] Richtwirkung: Frontal eintreffender Schall wird besser aufgenommen. Hier irrelevant.

kommen. Zur Veranschaulichung kann hier stark vereinfachend der Wechsel von Luft zu Wasser betrachtet werden [5, S.126]:
Luft (20°C):

$$Z_{Luft} = Z_1 = \rho_0 * c = 415 \ \frac{kg}{m^2 s}$$

Wasser (20°C):

$$Z_{Wasser} = Z_2 = \rho_0 * c = 1.48 * 10^6 \ \frac{kg}{m^2 s}$$

$\Rightarrow r = \frac{Z_2 - Z_1}{Z_2 + Z_1} = 0.9994$

Trotz dieser Vereinfachung betrüge der Verlust von Schallenergie durch Reflexion bei direkter Übertragung tatsächlich mehr als 90%. Die bereits genannte *Impedanzanpassung* des Mittelohrs mindert diesen Verlust, zum einen mechanisch über die Hebelwirkung der Gehörknöchelchen, als auch – zu einem größeren Anteil – über die Flächenverhältnisse[9] von Trommelfell (ca. 50mm^2) und ovalem Fenster (ca. 4mm^2). [7, S.680f]

Das hinter dem ovalen Fenster liegende **Innenohr** besteht aus der bereits genannten Gehörschnecke und dem Gleichgewichtsorgan, welches hier jedoch nicht weiter von Interesse ist. Die Schnecke liegt in einer spiralfömigen Einlassung direkt im Knochen (Felsenbein). Schwingungen, die über das Mittelohr auf das ovale Fenster verstärkend übertragen werden, lösen in der *Vorhoftreppe* (*Scala vestibuli*) eine stehende Welle aus, die sich über die inkompressible Flüssigkeit in Richtung des im Zentrum der Spirale liegenden *Schneckenlochs* fortbreitet, und über die *Paukentreppe* (*Scala tympani*) wieder zurück zum Mittelohr wandert, wo über das *runde Fenster* die Verdrängung der Flüssigkeit ausgeglichen wird. Die genaue Stelle dieser Druckübertragung von Vorhoftreppe auf Paukentreppe hängt von der Frequenz der Welle ab[10]. [7, S.683]

Zwischen den beiden "Treppen" liegt die *Scala media*, in welcher sich die für die Erzeugung von Nervenimpulsen verantwortlichen *Haarzellen* befinden. Begrenzt ist dieser innere Schlauch durch die *Reissner-Membran* (s. vestibuli - s. media) und die *Basilarmembran* (s. tymapni - s. media).[11] Eine Auslenkung der Letzteren versetzt dabei die Haarzellen ebenfalls in Bewegung.

Die sich entlang des Verlaufes verändernde Eigenfrequenz der Basilarmembran[12] ist es, die die zuvor genannte Frequenz-Abhängigkeit der Position des Druckaus-

[9] $\widetilde{p} = \frac{F}{A}$; Für $A_2 < A : \widetilde{p}_2 = \frac{F}{A_2} > \widetilde{p}$ (Druckverstärkung)
[10] Vgl. Abb.8, S.32
[11] Vgl. Abb.9, S.32
[12] Schmal und elastisch nahe des Mittelohrs, breit und unelastisch nahe des Schneckenlochs

gleichs verursacht.[13] Es findet also über diesen Mechanismus eine Frequenz-Ort-Transformation der Schallinformationen statt, welche über die durch die Haarzellen ausgelösten Impulse eine Unterscheidung der Frequenzen ermöglicht. [7, S.683f] Es können so also die einzelnen Komponenten einer überlagerten Welle aufgeschlüsselt werden.

Der Bereich der wahrnehmbaren Frequenzen hängt von der Anatomie des gesammten Ohres ab: Durch die Eigenfrequenz des Gehörgangs werden, bei der Schallleitung an das Trommelfell, Frequenzen im Bereich von $3kHz - 4kHz$ verstärkt [8, S.59]. Das Trommelfell selbst verstärkt, über seine Eigenfrequenz, Frequenzen zwischen $100Hz - 1.5kHz$ [10]. Die Impedanzanpassung im Mittelohr verursacht eine Schalldruckerhöhung um ca. $30dB$ [8, S.59]. Diese Größen lassen sich auch in empirischen Messungen des menschlichen Hörbereichs wieder erkennen (Abb.10, S.33).

Die Wahrnehmbarkeit von Lautstärke ist Resultat der vom Schallwechseldruck abhängigen variablen Entladungsfrequenz (Aktionspotentiale/Zeit) der inneren Haarzellen, wobei der wahrnehmbare Bereich so groß ist, dass er nur über ein logarithmisches Verhalten gänzlich umfasst werden kann[14]. Die Bewegung der Haarbündel wird dabei über eine durch Kompression verursachte Freigabe von Ionen und die folgliche Veränderung des zellineren Potentials, sowie eine daraus resultierenden Emission von Neurotransmittern, in eine Erregung der Rezeptoren der afferenten Neuronen gewandelt [11, S.308, 7.2.3]. Die Variabilität der Entladungsfrequenz ist dadurch gegeben, dass jede der Haarzellen durchschnittlich über 20 exklusive afferente Neuronen verfügt, welche jeweils unterschiedliche Erregungsschwellen besitzen. Bei zunehmendem Schalldruck werden immer mehr Schwellwerte überschritten und die Anzahl der feuernden Neuronen steigt. [7, S.688]

Das Zusammenspiel aus *äußeren Haarzellen* und *inneren Haarzellen* ermöglicht es, selbst leise Geräusche[15] wahrzunehmen. Im Gegensatz zu den Inneren, gehen von den Äußeren keine Nervenbahnen zum Gehirn aus[16]. Ihre Hauptaufgabe ist es, geringfügigere Verdrängungen in der Flüssigkeit zu verstärken, und so eine erhöhte Auslenkung der inneren Haarzellen zu verursachen. Dies ist möglich, da die Basilarmembran, von der die ursprüngliche Verdrängung auf die Scala media übertragen wird, im Bereich der äußeren Haarzellen flexibler ist (stärkere Schwingung) als im Be-

[13] Maximale Auslenkung der Basilarmembran bei $Eigenfrequenz = Wellenfrequenz$

[14] Vgl. hier die Definition des Schalldruckpegels [dB] in 2.1 und Abb.10, S.33

[15] geringe Amplitude der Schallwelle, bzw. geringer Schallwechseldruck

[16] efferente Fasern (vom zentralen Nervensystem aus), statt Afferenten (zum z. N.), wie bei Inneren; Funktion der efferenten Nervenbahnen die zu den äußeren Haarzellen führen nicht geklärt. Vermutlich zum Hemmen bei Hintergrundlärm [7, S.688]

reich der innen liegenden Haarzellen.[17] Der Verstärkungseffekt spielt bei von sich aus stärkeren Drücken allerdings kaum eine Rolle, und ist im Verhältniss zur entsprechend stärkeren Auslenkung der Basilarmembran vernachlässigbar gering. [8, S.60]

2.2.2 Gehirn

Sobald Frequenz und Amplitude der Schallwelle über das Innenohr als Nervenimpuls aus einem bestimmten Bereich mit bestimmter Feuerrate codiert ist, müssen diese Signale vom Gehirn noch zum Empfinden von Ton und Lautstärke interpretiert werden. Dabei werden bis hin zum auditiven Kortex mehrere Stationen[18] durchlaufen, welche bereits erste Schritte der Informationsverarbeitung umsetzen. Über die von den Hörschneken wegführenden (afferenten) Fasern, welche sich jeweils zum Hörnerv (*Nervus Cochlearis*) vereinigen, werden die Impulse bis zu den Schneckenkernen (*Nuclei Cochlearis*) im Hirnstamm geleitet. Diese sind jeweils weiter unterteilt in einen *dorsalen* und einen *ventralen* Kern, welche die Signale zu unterschiedlichen Stellen weiter geben. Eine erste Mustererkennung wird hier ebenfalls durchgeführt. Ausgehend vom ventralen Schneckenkern, erreichen die Impulse die *obere Olive* (*Nucleus olivaris superior*), wo die zeitliche Verzögerung, sowie Intensitätsunterschiede, zwischen den gelieferten Schallinformationen aus beiden Ohren analysiert wird [19]. Anschließend werden in den *unteren Hügeln* (*colliculus inferior*, im Mittelhirn), als Teil des *auditiven Reflexzentrums*, Schallinformationen beider Ohren zur geräuschabhängigen Steuerung von Körperbewegungen genutzt. Aus den unteren Hügeln werden die Signale wiederum zu einem der *mittleren Kniehöckern* (*Corpus geniculatum mediale*) geleitet, welche jeweils als eines der im Thalamus liegenden Schaltzentren für Sinnesinforationen dienen. Über die *radiato acustica* gelangen die verschalteten Impulse letzten Endes in die primäre Hörrinde, wo wiederum starke neuronale Vernetzungen über mehrere spezialisiertere Ebenen anzutreffen sind. Insgesamt lässt sich erkennen, dass hier – sogar bereits vor der eigentlichen Hörrinde – eine Verarbeitung der eingehenden Signale über vielfache Verschaltung umgesetzt ist. [7, S.689ff, 19.4.2] [7, S.690, Tabelle 19.4] [11, S.312ff]

2.2.3 Psychoakustik

Psychoakustik ist die Wissenschaft die sich mit dem menschlichen Hörsystem als Empfänger von akustischer Information, mit Fokus auf das Verhältnis zwischen Schall

[17] Vgl. auch hier Abb.9, S.32
[18] Vgl. im Folgenden auch Abb.11, S.34
[19] Lokalisierung von Geräuschquellen, hier wieder nicht von Interesse

und Empfinden, beschäftigt. [9, VII] Die wesentlichen Erkenntnisse für den Schalleitungsmechanismus, den das Außen- und Mittelohr implementieren, wurden bereits in 2.2.1 beschrieben. Für die anschließende Transduktion lässt sich jedoch noch einiges vertiefen.

Besonders interessant ist an dieser Stelle das Phänomen der Maskierung. Als intuitives Beispiel kann hier der Vergleich von Gesprächen in einem ansonsten ruhigen Zimmer, mit Unterhaltungen während eines Konzerts betrachtet werden: Der Gesprächspegel muss von dem Level des Zimmers drastisch angehoben werden, um nicht von der wesentlich lauteren Musik übertönt (maskiert) zu werden. Erklären lässt sich dieser Effekt unter anderem durch die *logarithmische* Abbildung der Schall-Amplitude über die Basilarmembran. Ein lautes Geräusch bewirkt eine wesentlich breitere Aktivierung der Fasern die für die maskierende Frequenz zuständig sind, als die der für die Maskierten Zuständigen.[(20)] Darüber hinaus gibt es noch weitere Maskierungseffekte, wie zum Beispiel die Post- und Pre-Stimulus-Maskierung, wobei Töne nach, bzw. vor, dem Auftreten des maskierenden Geräusches unterdrückt werden. Diese Prozesse sind nicht vollständig verstanden, es handelt sich jedoch vermutlich um das Verschwinden des im Vergleich "unwichtigeren" Geräusches im Unterbewusstsein. [9, S.61ff]

Wie beim Schallleitungsprozess sind auch hinter dem Transduktionsaparat die wesentlichen Erkentnisse bereits in der anatomischen Beschreibung (2.2.2) erfasst: Die rund 30.000 afferenten Nervenfasern, die – wie in 2.2.1 beschrieben – exklusiv und gebündelt von den Haarzellen ausgehen, erhalten zwangsweise zunächst die frequenzielle Partition des auditorischen Inputs. Dies wird auch als *Tonopie* bezeichnet. Tiefer liegende Ebenen des dahinter aufsteigenden neuronalen Netzes integrieren Informationen aus beiden Ohren für unterbewusste Vorgänge – wie zum Beispiel für Lokalisierung – und höhere Ebenen werden zunehmend spezialisierter und komplexer. Hier fehlt es im allgemeinen allerdings auch noch an Verständnis für genaue Aufgaben und Funktionsweisen. [9, S.58ff]

Im Rahmen der Psychoakustik kann zudem die komplexe Empfindung von Musik weiter charakterisiert werden. Als catch-all für Eigenschaften außerhalb von Lautstärke und Tonart dient der Begriff *Timbre*, auch *Tonfarbe*. Als erste Veranschaulichung für das Konzept hinter Timbre, und warum in diesem Begriff so viele Faktoren zusammenlaufen, kann der Vergleich von verschiedenen Instrumenten dienen. Eine Gitarre und eine Klavier klingen unterschiedlich, obwohl sie den selben, über eine Frequenz eindeutig definierten, Ton spielen. Die Tonfarbe lässt sich (in der perzeptuellen Wirkung nicht ganz leicht) unter anderem in die Obertöne, welche den als Ton wahregenom-

[(20)] Vgl. 2.2.1, den Abschnitt um Fußnote (14)

menen Tonkomplex bilden, spektrale Komposition (Hüllkurve), Schärfe und *sensory pleasentness*[21] aufgliedern. Eine intuitive und zugleich präzise Definition von Timbre gibt es jedoch nicht. An dieser Stelle soll es reichen zu erkennen, dass das Fassen der musikalischen Empfindung äußert komplex ist, und die bisher verwendeten Merkmale vielfältig ergänzt werden können. [12] [9, S.239ff]

2.3 Musik als Datengrundlage

Die Tatsache, dass Computer sowohl zeit- als auch wertdiskret arbeiten, macht eine Abbildung von zeit- und wertkontinuirlichen Schallsignalen nicht-trivial. Es müssen in jedem Fall Kompromisse gefunden, Vereinfachungen angestellt, und Informationsverluste in Kauf genommen werden. Da eine maschinelle Verarbeitung von Musik Daten in digitaler Form voraussetzt, ist es sinnvoll, an dieser Stelle die nötigen Grundlagen der unterliegenden Abbildung zu klären und so ein Verständnis für die Struktur dieser Daten zu erlangen.

Es ist dabei unabdinglich zu erwähnen, dass diese Codierung eine äußerst komplexe Angelegenheit darstellt, und dem entsprechend im Folgenden – dem Umfang dieser Arbeit geschuldet – nur oberflächlich betrachtet werden kann. Wir geben uns also damit zufrieden, die prinzipiellen Vorgänge, die für unser Projekt relevanten Datei-Formate, sowie die bereits in der Codierung enthaltenen Modellierungen und Berücksichtigungen des menschlichen Hörens zu verstehen.

2.3.1 PCM und Kenngrößen

Das im Rahmen der *Pulse Code Modulierung* umgesetzte *Codieren* von analogen (kontinuirlichen) Audio-Signalen zu Digitalen erfolgt grundsätzlich in zwei Phasen: Zunächst wird die Amplitude der Welle in diskreten regulären Zeitabständen abgetastet (*gesampelt / Sampling*) und dann auf einen ebenfalls diskreten Wertebereich abgebildet (*Quantisierung / Quantization*). Zur Wiedergabe von digitalen Audioquellen, muss die selbe Prozedur in die andere Richtung gespielt werden. Das *Decodieren* besteht aus der Umkehrung der Quantisierung und dem *Interpolieren*. Aus den diskreten Werten wird so die ursprüngliche kontinuirliche Welle wieder angenähert. Je nach Feinheit der Quantisierung (Sampling Frequenz $[kHz]$ und Bit-Tiefe) werden so im Endeffekt mehr oder weniger Informationen verloren. [22]

Die "Feinheit" kann dabei über die *Sampling Frequenz* $[kHz]$ und die *Bitdepth*

[21] Wiederum zusammengesetzt aus weiteren Aspekten
[22] Vgl. auch Abb.12, S.35 und Abb.1, S.9

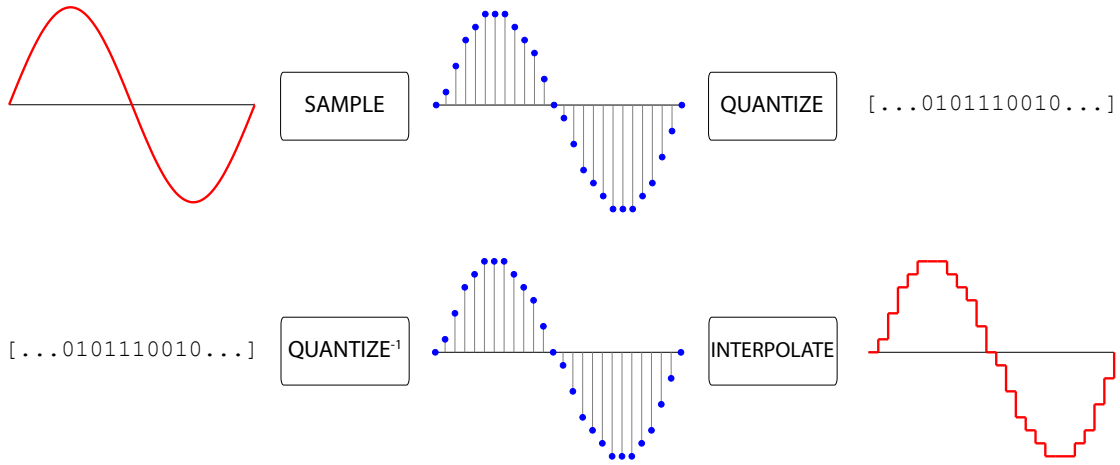


Abbildung 1: PCM Codec. Encoder (oben) und Decoder (unten). Basierend auf [13] und [14, S.7, Fig.4]

$[Bit]$ definiert werden[23]. Die Sampling Frequenz gibt an, wie viele Samples pro Sekunde erhoben werden und bestimmt welche Frequenzen aufgezeichnet werden können[24]. Die Bitdepth gibt an, wie viele Bit pro Sample zur Verfügung stehen. Bei einer Tiefe von 16 Bit, gibt es zum Beispiel $2^{16} = 65.536$ mögliche Stufen, womit ein Lautstärkenbereich von 90 Decibel umfasst werden kann. [14, S.7f]

Der Prozess der Quantisierung ist dabei in der Regel die Hauptquelle von, in Folge der Codierung entstehenden, Verzerrungen und Rauschen (*noise*) [14, S.13]. Wie dieser bei der Diskretisierung entstehende Informationsverlust unter Berücksichtigung der Perzeption des Menschen "reduziert" werden kann, wird in 2.3.2 bei der Betrachtung von MP3 beschrieben.

Gemessen wird die Qualität der Quantifizierung über die sogenannte *Signal to noise ratio (SNR)* $[dB]$, welche als skaliertes Verhältnis der Leistung[25] von Inputsignal $(x_{in}(t))$ zur Differenz der Leistung von Output- $(x_{out}(t))$ und Inputsignal (*Fehlersignal* $q(t)$) definiert ist:

$$\left(SNR = 10 * log_{10} \frac{< x_{in}(t)^2 >}{< (x_{out}(t) - x_{in}(t)) >^2} = 10 * log_{10} \frac{< x_{in}(t) >^2}{< q(t) >^2}\right) \ [dB].$$

Je geringer das Fehlersignal, desto größer die SNR. [14, S.34]

Eine reduzierte SNR kann generell auf zwei Fehlerquellen zurückgeführt werden:

[23] *Bitrate* = #Kanäle ∗ Sample-Rate(/Sampling Frequenz) ∗ Bitdepth(/Auflösung); auch *Datenrate*

[24] Zur Ertastung einer Frequenz f muss die Sampling-Frequenz mehr als doppelt so groß sein wie f; Samplingtheorem [15, S.34, Theorem 13]

[25] Leistung wird hier wie in [14] mit "<>" notiert.

Rundungsfehler (*round-off error*) und *Clippingfehler* (*clipping / overload error*). Rundungsfehler sind dabei offensichtlich abhängig von der Bittiefe, die zur Diskretisierung der Amplitude bereit gestellt wird: Je weniger Stufen zur Verfügung stehen, desto breiter die Bereiche die auf den selben Wert abgebildet werden (Rundung) und auf diese Weise größere Abweichungen (Fehler) verursachen. Für einen uniformen Quantiserer mit Sinuswelle als Input lässt sich vereinfachend zeigen, dass für jedes Bit welches zur Speicherung hinzugenommen wird, sich die SNR um etwa 6 Decibel erhöht. [14, S.35f]

Clippingfehler entstehen bei einem zu niedrigen Maximalwert (/ zu hohem Minimalwert) der Amplitudenabbildung, bzw. bei zu hohen (/ zu niedrigen) Amplituden des Inputsignals. Diese Werte werden *geclippt*, und alle Informationen über (/ unter) dem Schwellwert gehen verloren. Hier ist bei dem Design eines Quantisierers eine Balance zu finden. Bei gleichbleibender Bittiefe benötigt eine weitere Amplitudenabdeckung größere Stufen, und verursacht so einen größere Rundungsfehler, und umgekehrt. Der wahrnehmbare Effekt von Clipping ist die Übersteuerung, Rundungsfehler resultieren in allgemeinem Rauschen[(26)]. [14, S.37]

Zur effizienteren Nutzung der verfügbaren Bit, kann sich auch hier der Codierung nach Entropiegehalt (genauer Huffman) bedient werden. [14, S.41ff]

2.3.2 WAV und MP3

Einer der am weitesten verbreitete Container für PCM Daten ist das, ursprünglich von Microsoft entwickelte, *Waveform Audio File Format* (*.wav*). So ist dieses der definierte Standard der europäischen Rundfunkunion (European Broadcasting Union, EBU) und vieler Audio-Archive. [18] Daten in dieser Form sind also, abhängig von der Sampling-Rate und der Bitdepth, als PCM-Wrapper dem analogen Signal am nächsten.

Nahezu allgegenwärtig ist jedoch, statt WAVE, die *MPEG Layer III Audio Encoding* (*.mp3*) [19]:

MPEG-1[(27)] ist ein Standard[(28)] für die Kompression von synchronisiertem Video und Audio, wobei der Audio-Teil der Spezifikation die Syntax des codierten Bitstreams, sowie den Decodierungsprozess enthält. Der Codierprozess ist nicht Teil des Standards, wird jedoch als Referenz beigefügt. [14, S.268] Unter MPEG-1 Audio werden drei verschiedene Codierungsprofile mit aufsteigender Komplexität und Audioqua-

[(26)] Siehe Beispielsweise: [16] und [17]
[(27)] Erweiterungen wie MPEG-2 verändern nicht den Kern der Arbeitsweise [14, S.315]
[(28)] seit 1992 in [ISO/IEC 11172] definiert

lität definiert: Layer I, Layer II und die stark veränderte Layer III.[29] Alle drei Layer unterstützen Sampleraten von 32kHz, 44.1kHz und 48kHz, mit jeweils unterschiedlichen Bitraten, wobei die für uns wichtige Layer 3, mit Datenraten zwischen 32kHz und 160kHz, etwas unter denen von Layer I und II liegt. [14, S.269f] Ohne weiter in die Details der Umsetzung zugehen, soll nun der vorgeschlagene Codierungsprozess dieser dritten Layer (MP3) skizziert werden:

Das Kompressionsverfahren hinter MP3 setzt, unter Verwendung psychoakustischer Konzepte, auf sogenannte *perzeptuelle Codierung*. Wie in 2.3.1 angedeutet, soll der wahrnehmbare Informationsverlust möglichst reduziert werden. Die grundlegende Idee ist dabei Frequenz- und Intensitäts-Bereiche großzügiger zu codieren, wenn diese vom Menschen besonders gut wahrgenommen werden können, und im Gegenzug mehr Verlust bei weniger gut wahrnehmbaren Bereichen zu akzeptieren. Verständlicher Weise ist es nötig, um sich dem menschlichen Gehör anzupassen, bestimmte Parallelen im Codierungsprozess umzusetzen. So wird sich in der mehrschrittigen Verarbeitung unter anderem einer Form der *Fourier Transformation* bedient – einer Abbildung des Signals aus der Zeitlichen in die Frequenzrepräsentation ($x(t) \rightarrow X(f)$) [14, S.51].[30]

Spezileller wird über PQMF (siehe 2.3.3) eine erste Zeit-Frequenz-Transformation mit 32-Bändern durchgeführt, welche anschließend durch eine dynamische MDCT Transformation weiter verarbeitet wird, um schließlich nicht-uniform und in Abhängigkeit von einem psychoakustischen Model quantisiert zu werden. Diese dynamische Abbildung über Berücksichtigung der Wahrnehmung von Musik ist es, was MP3 erlaubt starke kompression mit geringem *wahrnehmbaren* Verlust umzusetzen.

2.3.3 Modellierung (Fourier & MEL)

Abschnitt 2.3 abschließend, soll noch die bereits mehrfach erwähnte Zeit-Frequenz-Transformation, als einfachste Annäherung an die vom Innenohr durchgeführte Ort-Frequenz-Transformation, betrachted werden.

Die *Fourier Transformation* beruht auf der
FT [14, S.51]
PQMF [14, S.75ff]
MDCT [14, S.103ff]
[14, S.51]
[14, S.51]

[29] Vgl. hier und im Folgenden Abb.13, S.35 und Abb.14, S.36
[30] Vgl. Ort-Frequenz Transformation; Genaueres zur Fourier Transformation im Abschnitt 2.3.3

2.4 Artificial Neural Networks

An dieser Stelle soll nun die grundlegende Idee hinter *künstlichen neuronalen Netzen* skizziert, und die verschiedenen Arten mit ihren Anwendungsgebieten betrachtet werden.

Mittels künstlicher neuronaler Netze wird versucht die Struktur von biologischen Neuronen zu modellieren, und durch Vernetzung die Kapazitäten des Gehirns für Mustererkennung in vereinfachtem Maße rechnerisch nutzbar zu machen. Grundeinheit solcher Netze sind offensichtlich einzelne künstliche Neuronen[(31)]:

Eingehende Signale $x_1, ..., x_m$ werden mit für das entsprechende Neuron N spezifischen Gewichten $w_{1N}, ..., w_{mN}$ multipliziert. Die Summe dieser gewichteten Signale wird als net_N bezeichnet:

$$net_N = \sum_{i=0}^{m} w_i * x_{iN} = X * W = (x_1, ..., x_m)^T * (w_{1N}, ..., w_{mN})^T$$

wobei $x_0 = 1$ (fest) mit w_{0N} den *bias* beschreibt. Die Aktivierungsfunktion bestimmt den Output y des Neurons als $f(net_N) = y$. Einfache Beispiele für Aktivierungsfunktionen sind *hard limits* (Schwellwert s), bei dessen Überschreitung ein konstanter Output z geschaltet wird, oder lineare Funktionen, bei denen der Output je nach Größe von net_N steigt:

$$f_{hard}(net_N) = \begin{cases} z, & if net_N \geq s \\ 0, & if net_N < s \end{cases} \qquad f_{lin}(net_N) = \lambda * net_N$$

[20, S.202f]. Diese Neuronen können nun mit einander Vernetzt werden, indem der Output als Input für andere Neuronen verwendet wird[(32)].

2.4.1 Arten künstlicher neuronaler Netze

Neuronale Netze lassen sich auf oberste Ebene, anhand des Verhaltens der Kanten, im wesentlichen in zwei Kategorien einteilen: Lässt sich das Netz in klare Schichten teilen – Die erste Schicht erhält den Input, die zweite den Output der ersten, usw. – und gibt es keine Kanten die aus einer tieferen Schicht in eine obere führt, so handelt

[(31)] Vgl. im Folgenden Abb.15, S.36
[(32)] Vgl. z.B. Abb.17, S.37

es sich um ein *feedforward* Netz. Gibt es jedoch diese rücklaufenden Kanten, also wird der Output einer tieferen Ebene als Input einer Früheren im nächsten Schritt verwendet, so handelt es sich um ein *rekurrentes* Netz. [(33)] [20, S.206f]

Ergänzend sind die Bezeichnungen *convolutional* und *deep* in einer Vielzahl von Architekurtypen anzutreffen. "Deep" gibt dabei lediglich Auskunft über die stark erhöhte Anzahl von Layern, "Convolutional Nets" verwenden hingegen drei verschiedene Arten von Layern: *convolutional*, *pooling* und *fully connected*. Die convolutional Layer mappen den Output der vorherigen Layer über eine Filtermatrix auf die nächste. Es werden so lokale Eigenschaften erlernt. Die pooling Layer reduziert die Dimensionalität der vorherigen Layer, und jedes Neuron einer fully connected Layer ist mit jedem Neuron der nächsten Ebene verbunden. [22, S.4f] Darüber hinaus gibt es mittlerweile eine Vielzahl an Kombinationen und Netzarten[(34)].

2.4.2 Training

Der Grund warum künstliche neuronale verwendet werden, ist das Versprechen, dass sie aus Daten lernen können. Im Falle von *supervised learning* werden dem auf das Problem angepasste Netz gelabelte Daten gegeben (Inputvektor X), und der Output wird mit den Labels verglichen. Hierzu wird eine *Error-Funktion* benötigt, welche die Abweichung vom gewünschten Ergebniss quantifiziert. Das eigentliche Lernen besteht nun darin, diesen Fehler durch Anpassung der Gewichte (W) zu minimieren, und so itterativ eine komplexe Funktion, welche implizit in der Struktur der Daten enthalten ist, anzunähern. Diese Minimierung geschieht nach in einem *Lern-Algorithums* definierten Regeln.

Allgemein lässt sich der Fehler eines Neurons k als die Differenz des gelieferten ($y_k(n)$), und des erwarteten Outputs ($d_k(n)$) zum aktuellen Zeitpunkt (n) definieren:

$$e_k(n) = d_k(n) - y_k(n)$$

Die Error-Funktion $E(n)$ kann über $e_k(n)$ beliebig defniert werden.

$$\Delta w_{kj}(n) = \eta * e_k(n) x_j(n)$$

definiert die Anpassung des Gewichts der Kante zwischen dem Neuron j der vorherigen Ebene und dem Neurons k mit dem Fehler e_k zum Zeitpunkt n und der *Lernrate*

[(33)] Vgl. Abb.16, S.36
[(34)] Vgl. [21]

η. Umgesetzt wird diese Anpassung im nächsten Schritt:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}$$

[20, S.207ff]

Für das gesamte Netz wird diese Anpassung von der Output-Layer bis zur Input-Layer über *Backpropagation* rückwärts durchgeführt.

3 Umsetzung

3.1 Grundlagen untersuchen

Auf Grund der Tatsache, dass wir zuvor noch nie, weder mit audio Daten, noch mit neuronalen Netzen gearbeitet hatten, allerdings ein starkes Interesse an der Thematik und der damit Verbundenen menschlichen Wahrnehmung hatten, legten wir zunächst einen starken Fokus auf die Erarbeitung der entsprechenden Kenntnisse. Unglücklicherweise geschah dies zunächst – entgegen entsprechenden Tipps der Betreuer – informal in diversen Notizblöcken und Text-Dateien. Das so bereits gewonnene Wissen musste im Rahmen dieses Berichts gegen Ende der Bearbeitungszeit also noch mühsam wieder aufarbeitet werden. Der Anspruch dabei, möglichst wissenschaftlich zu arbeiten, erforderte es zahlreiche Stunden in die Suche und Aufnahme seriöser Quellen zu investieren. Da es sich hierbei jedoch um das erste und grundlegende Ziel unseres Projektes handelte, und der Bericht ein wesentliches Kriterium der Bewertung darstellt, entschlossen wir uns, diese Arbeit zu priorisieren.

Besonders im Bereich der Anatomie und der Psychoakkustik war es schwer, Quellen außerhalb der Wikipedia zu finden, da die meisten Bücher in diesen Gebieten über 100 Euro kosten. Glücklicherweise fand sich nach langer Suche jedoch immer ein Weg, die benötigten Texte einzusehen.

3.2 Suche nach Daten

Als wir mit der Umsetzung unseres Projekte begannen stellte sich heraus, dass die Suche nach einer Methode, um große Mengen an Daten in Form von Musikdateien zusammenzutragen, eine größere Herausforderung war als erwartet. Diese Phase der Umsetzung beschäftigte uns einen Großteil der Projektdauer.

Es galt eine Methode zu finden, mit der es möglich ist, möglichst automatisiert und mit wenig manueller Arbeit '*geeignete Musikdateien*' anzusammeln. Der Faktor der Automatisierung ist wichtig, weil wir vor haben einen möglichst großen Datensatz aufzubauen, weil wir davon ausgehen, dass unsere Analysen und deren Präzision für unsere Anwendung der Genre-Klassifizierung linear mit der Größe der Datengrundlage steigen werden.

Die Formulierung '*geeignete Musikdateien*' ist hier bewusst uneindeutig gewählt, weil sie von uns selber zu Beginn der Suche nach einer Methode zur Sammlung von Musik noch nicht vollständig und eindeutig definiert werden konnte. Die gewünschten Kriterien und Voraussetzungen an den Datenpunkten und dem gesamten Datensatz

als solchem, die diesen für unseren Zweck geeignet machten, ergaben sich in unserer Projektarbeit erst in einer inkrementellen Weise nachdem wir mehrere Methoden betrachtet, in Erwägung gezogen, implementiert und ausprobiert haben. Im Folgenden möchten wir diese Prozess schildern, um dazulegen, wie sich unsere finale Wahl der Methode herauskristallisiert und die Struktur und Beschaffenheit unseres Datensatzes ergeben hat.

3.2.1 FMA

Eines der größten Probleme, welche sich bei der Datensammlung ergeben hat, ist urheberrechtlicher Natur. Herkömmliche Musik ist mit dem Urheberrecht geschützt und wird von den Rechteinhaber meistens kommerziell verwertet und verkauft. Die Option, dass wir uns einen Datensatz zusammenkaufen, erschien uns, hinsichtlich unserer anvisierten Datensatzgröße und unserer finanziellen Möglichkeiten, nicht im Rahmen des Möglichen. Wir selber besitzen privat eigene Musiksammlungen, welche wir über Jahre angesammelt haben. Diese sind uns aber wegen unseres eingeschränkten persönlichen Geschmacks zu schmal in der Genreabdeckung und davon abgesehen sind unsere Sammlungen zu wenig strukturiert, als dass wir sie ohne großen manuellen Aufwand für unsere Analyse benutzen könnten.

Unser Fokus änderte und richtete sich deshalb an die Musikszene mit Lizenzfreiheit, welche im Internet dank Bewegungen wie *"Creative Commons"* eine große Verbreitung hat. Es gibt verschiedene Plattformen, die, in unterschiedlich starken Modellen, günstige oder lizenzfreie Musik zur freien Verwertung und Benutzung zu Verfügung stellen. Auch hier mussten wir aber natürlich den Punkt der Automatisierung berücksichtigen. Nach der Untersuchung verschiedener Plattform entdeckten wir das *"Free Music Archive"*[23] als erfolgversprechendste Website. Die Plattform bietet Musikstücke als MP3 Dateien an. Die Musikstücke sind Genres zugeteilt, über welche diese in einer Listensicht einsehbar sind. Das FMA bietet allerdings keine Möglichkeit eine Liste mit allen Musikstücken auf einmal herunterzuladen. Für unseren Zweck müssten wir also jede Musikdatei einzeln manuell heraussuchen und herunterladen, was zu viel Aufwand wäre. Glücklicherweise entdeckten wir, dass im Sourcecode der Detailseiten der einzelnen Musikstücke die URLs zur MP3 Datei eingebettet war. Mit diesem Wissen gelang es uns einen Crawler in Python zu implementieren, mit welchem wir automatisiert viele Musikstücke eines Genres mit einem mal herunterladen konnten.

Nachdem wir über diesen Weg eine Zeit lang angefangen haben unseren Datensatz aufzubauen untersuchten wir die heruntergeladene Musik genauer. Bei dieser

Untersuchung stellten wir fest, dass uns die Qualität der Musik nicht ausreichte. Das FMA wird zu einem große Teil dafür benutzt lizenzfreie Hintergrundmusik für Videoproduktionen und Kurzfilme anzubieten. Diese Hintergrundmusik ist qualitativ auf einem anderen Niveau als kommerzielle Musik und die Genreeinteilung ist nicht besonders akkurat. So ist uns aufgefallen, dass Genres wie Hip-Hop oder Jazz im FMA sehr offen und weit definiert werden und zum Teil experimentelle und alternative Interpretationen dieser Genres häufiger vorkommen als Musikstücke, welche unserer Vorstellung jenes Genres entsprechen. Auch sind diese Stücke im Aufbau grundlegend anders als herkömmliche Musikstücke, da die Titel vom FMA vermutlich selten als ganzes in Videoproduktionen verwendet werden sondern nur Ausschnitte von diesen. Häufig sind die Musiktitel auch ohne vokale Komponente, sodass wichtige genrespezifische Eigenschaften wie Gesangsstil und Verse-Refrain Struktur mit der fehlenden Gesangsstimme komplett unberücksichtigt bleiben würden in unserem Klassifizierungsmodell, wenn dieses Musik aus dem FMA als Trainingsgrundlage nutzen würde. Hinzu kommt noch, dass die dort angebotene Musik häufig von unbekannteren Indie-Interpreten in Eigenregie veröffentlicht wird, sodass uns für eine mögliche, weitergehende Analyse vermutlich keine Daten wie Songtexte oder Metadaten von Titel und Interpret zur Verfügung stünden über Datenbanken wie *"MusicBrainz"* [24] oder *"MusiXmatch"*[25].

Aufgrund dieser Gegebenheiten und neu hinzugekommenen, uns wichtig erscheinenden, Anforderungen an die Musikstücke mussten wir diese Art der Musikbeschaffung aufgeben.

3.2.2 Spotify

Nachdem wir entdeckten, dass es eine Reihe von Open Source und kommerziellen Diensten gibt, die sich explizit damit befassen, Metadaten und Songtexte zu Musiktiteln zur Verfügung zu stellen, durchsuchten wir das Internet nach weiteren Projekten, Angeboten und Diensten, welche uns in der Datenbeschaffung nützlich sein könnten. Bei dieser Suche stoßen wir häufig auf den Namen *"libspotify"*. Bei der libspotify handelt es sich um eine Bibliothek vom Musikstreaming-Dienst Spotify, auf der aufbauend viele quelloffene Projekte auf GitHub existieren. Dabei handelt es sich um eine Bibliothek, welche es Entwicklern ermöglicht eigene Client Implementationen für Spotify zu schreiben. Die Bibliothek kommuniziert dabei direkt mit dem Spotify-Backend und ermöglicht so das Abrufen und Streamen von Musiktiteln. Wir entdeckten einige fertige Projekte, die es ermöglichen gestreamte Musik zu rippen, als lokal auf der eigenen Festplatte als Audiodatei zu persistieren.

Nach eigenen Angaben sind über Spotify 35 Millionen Titel verfügbar[26]. Wir beide benutzen den Dienst auch privat und wussten, dass die Auswahl von Titeln sehr hochwertig ist und unserem Zwecken definitiv genügen würde. Es existieren viele, von Nutzern und professionellen Musik-Kuratoren erstellte, Playlisten, von denen sich einige einzelnen Genres widmen

Über einen der auf libspotify aufbauenden Ripper wäre es uns also für unseren Zweck möglich einen Musikdatensatz zu erstellen, welcher dank der Genre-Playlisten nach Genres sortiert wäre. Der Zugriff auf die libspotify wird aber von Spotify über vorher ausgegebene API-Keys kontrolliert. Leider mussten wir feststellen, dass Spotify die libspotify als deprecated eingestuft und das Formular zur Beantragung eines API-Keys offline genommen hat. Über verschiedene Foren erfuhren wir aber, dass Spotify wohl noch vereinzelnd Keys ausgibt, weshalb wir uns entschieden Spotify in einer Email und einem Brief von unserem Vorhaben zu schildern Musik temporär zu speichern, in ein Spektrogramm umzuwandeln und anschließend wieder zu löschen, um so einen API-Key zu bekommen. Leider haben wir nach 2 Wochen keine Antwort bekommen, sodass wir uns nach neuen Möglichkeiten umschauen musste.

Dabei entdeckten wir, dass Spotify gerade ein neues Android-SDK [27] als eingeschränkten Nachfolger für die libspotify mit weniger Funktionen entwickelt. Dieses SDK befand sich zum Zeitpunkt unseres Projektes noch in einer frühen Beta-Phase. Wir verfolgten den Entwicklungsstand und entdeckten, dass in einem der Beta-Release ein neues Java-Package namens *"SpotifyPlayer"* hinzugekommen ist. Zu diesem Zeitpunkt war das Package und seine Funktion noch undokumentiert. Wir beschäftigten uns mit der API des *SpotifyPlayer* im SDK und untersuchten den Netzwerkverkehr unseres Testgerätes. Nach einigem Ausprobieren fanden wir heraus, dass über den *SpotifyPlayer* Musik gestreamt und wiedergegeben werden kann. Über eine undokumentierte Methode gelang es uns einen eigenen Audiocontroller in den *SpotifyPlayer* zu integrieren. Dieser wird mit PCM-codierten Daten aufgerufen und ist für die Wiedergabe der Musik zuständig. Auf Android konnten wir keine Audio-Bibliothek einbinden, sodass wir diesen PCM-Stream selber behandeln mussten. Nach intensiver Beschäftigung mit dem .WAV Format und dessen Formatierungsdetails gelang es uns gestreamte Titel als .WAV Datei auf unserem Android-Emulator aufzuzeichnen.

Um aber über diese Methode automatisiert unseren Datensatz aufbauen zu können mussten wir uns zusätzlich noch Gedanken machen. Unsere Idee war es mehrere Instanzen des Android-Emulators mit unserem *"Rippify"* getauften Ripper parallel laufen zu lassen und diese jeweils verschiedene von uns ausgewählte Playlisten rippen zu lassen. Das Übertragen der gerippten Musik aus dem Container der Emulatoren auf

unsere lokale Festplatte erwies sich aber als überaus schwierig. Der Emulator verwendet als Speichermedium eine simulierte SD-Karte, welche in Form eines .iso Images auf der Festplatte liegt. Uns gelang es nicht dieses Image auszulesen und die gerippte Musik aus diesem zu extrahieren. Das Android-Projekt stellt noch einen *"android device bridge(adb)"* zur Verfügung, über welche Daten zwischen dem Filesystems des Emulators und dem Host-Computer ausgetauscht werden können. Bei Dateien größer als 4 MB oder dem Transfer von ganzen Verzeichnissen funktionierte die *adb* bei uns aber nicht.

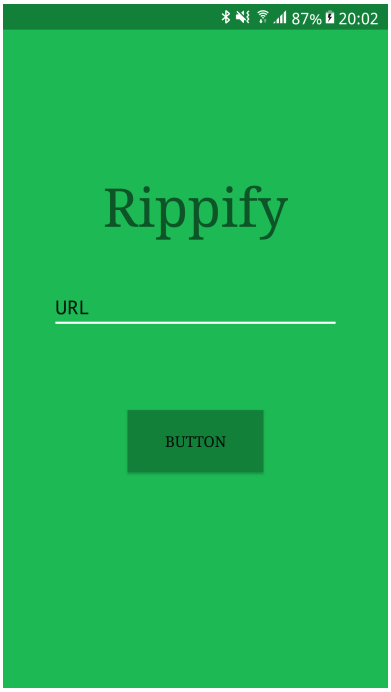


Abbildung 2: Screenshot der Rippify Android App

Diesen Schwierigkeiten kam noch hinzu, dass das Android-SDK in einer frühen Betaphase war und deshalb keineswegs verlässlich funktioniert hat. Die Authentifizierung, welche vor dem Streaming getätigt werden muss, schlug unregelmäßig aus unbestimmten Gründen immer mal wieder fehl. Das Verfahren mit mehreren Emulatoren war zudem sehr ressourcenaufwendig und das Aufteilen der Netzwerk-Bandbreite über mehrere Emulatoren sehr ineffizient. Auch war uns der rechtliche Aspekt sehr unsicher, da aus den Nutzungsbedingungen des SDKs keine Einschränkungen bezüglich der temporären Persistierung von Audiostreams vom *SpotifyPlayer* existieren, wir uns aber vorstellen konnten, dass eine solche Nutzung nicht im Interesse von Spotify ist. Da wir ohnehin nicht unsere gerippte Musik aus den Emulatoren heraus bekommen konnten mussten wir auch dieses Verfahren nach vielem Ausprobieren und Herum-

experimentieren aufgeben.

3.2.3 Internet-Radio

Nach vielem überlegen bekamen wir den Tipp uns mit Internet-Radio zu beschäftigen. Nach einer Recherche entdeckten wir, dass es eine sehr große Auswahl an kostenlos zu empfangenen Stationen im Internet gab. Neben Städtestationen und Talkstationen gibt es eine große Auswahl an Musikstationen, von denen sich einige dediziert einzelnen Genres widmen. Bevor wir uns Gedanken machten wie wir hier die Musik rippen können recherchierten wir die rechtliche Lage und fanden heraus, dass es in Deutschland den rechtlichen Begriff der Privatkopie gibt [28], welcher uns erlaubt legal Aufnahmen und Mitschnitte vom Radio und auch vom Internet-Radio zu machen solange wir mit diesen Aufnahmen nicht handeln oder anders kommerziell verfahren.

Jetzt, wo die rechtliche Sicherheit gegeben war, begannen wir einen Ripper für Internet-Radio in Python zu schreiben. Die Herausforderung hier war es den Audiostream an den passenden Stellen zu unterbrechen, wenn die Lieder sich bei der Station wechselten, um hinterher einzelne Audiodateien zu jeweils einem Musiktitel zu haben. Über Internet-Radio Clients konnten wir beobachten, dass über den Stream wohl auch Metadaten wie Titel und Interpret übertragen werden. Wenn es uns gelingen würde diese Metadaten zu empfangen und auszuwerten könnten wir an den Zeitpunkten, an denen diese wechseln, einen Cut setzen und so einzelne Audiodateien rippen.

Die von uns betrachteten Internet-Radio Stationen verwenden das ICY Protokoll. Dieses, von SHOUTcast entwickelte, proprietäre Protokoll ist aber sehr schlecht dokumentiert, sodass wir Mühe hatten diese Metadaten auszulesen.

Glücklicherweise entdeckten wir zu diesem Zeitpunkt die C geschriebene Software "Streamripper", die perfekt für unseren Anwendungsfall war. Über die Kommandozeile kann man dem Streamripper einen Link zu einer Station als Input geben und der Streamripper übernimmt von da an das Lesen des Audiostreams inklusive der Metadaten und der Separierung einzelner Musiktitel. Diese werden während des Rippens auf die Festplatte als .MP3 Dateien gespeichert und nach Interpret und Titel benannt.

Mit diese Methode war es uns also möglich legal und verlässlich Musikdateien zu speichern. Über die Auswahl der Stationen konnten wir genrespezifisch arbeiten und die Qualität der Musik (bekannte Titel, keine allzu experimentelle Interpretationen der Genres usw.) grob steuern. Zusätzlich konnten wir so auch noch die Audioqualität in Form von verschiedenen Bitraten ausgleichen über die Auswahl von entsprechen-

den Stationen, sodass wir bspw. zu eine Genre nicht nur Musik minderer Qualität als Trainingsgrundlage haben und nachher in der Genre-Klassifizierung mit einem hochqualitativem Titel Probleme bekommen. Der Streamripper arbeitet ressourcenarm, sodass wir mehrere Stationen gleichzeitig rippen konnten je nach dem, wie viel Bandbreite uns zur Verfügung stand.

Nachdem wir Musikstationen nach unserem Kriterien an Qualität der gestreamten Musik zusammengestellt hatten haben wir uns dafür entschlossen, uns auf die drei Genres Jazz, Klassik und Rock zu beschränken. Grund für diese Entscheidung war auch, dass wir glauben, dass diese drei Genres wohl unterscheidbar voneinander sind und eine Einschränkung auf diese eine Vereinfachung für unser Klassifikationsproblem sein wird. Als gewünschten Richtwert der Datenmenge haben wir uns auf 10 Gigabyte pro Genre festgelegt, auch wenn wir in der Praxis bei einigen Genres aufgrund von Mangeln an guten Stationen und Songdopplungen nicht auf diese Zahl gekommen sind. Die Entscheidung des Richtwerts bei 10 Gigabyte ist pragmatischer Natur und ist gefallen, nachdem wir erwartete Genauigkeit des Klassifikators mit erwarteter Trainingsdauer und Rechenintensität abgewägt haben.

Übersicht über unseren Datensatz:

Tabelle 1: Verwendete Radiosender

Genre	Sender	|Songs|	Bitrate[35][36]	|PNG|[37]
Classic	AbacusFM Mozart	299 (1.70GB)	128	-
Classic	Audiophile Classical	82 (3.82GB)	320	-
Classic	Classical Connection on Radio 257	271 (3.37GB)	256	-
Classic	Venice Classic Radio Italia	171 (1.67GB)	128	-
$\sum$Klassik	-	823 (10.56GB	-	42020 (3.37GB)
Jazz	Audiophile Jazz	177 (2.11GB)	320	-
Jazz	AW Music	80 (0.82GB)	320	-
Jazz	DI Radio Digital Impulse - Jazz	123 (1.10GB)	320	-
Jazz	Jazzgroove	170 (0.75GB)	128	-
$\sum$Jazz	-	550 (4.78GB)	-	15118 (1.25GB)
Rock	Its time for rock'n roll	72 (0.14GB)	128	-
Rock	Psychadelic FM	113 (0.38GB)	128	-
Rock	Rock Radio Beograd (+comericals)	94 (0.35GB)	128	-
$\sum$Rock	-	279 (0.86GB)	-	5526 (0.50GB)

[35] $[kB/s]$; Vgl. 2.3.2: Max. Bitrate von 32-160kHz *pro Kanal*; stereo → max. von 320kHz

[36] Alle PCM Daten vor der Generierung zu Spektrogrammen geresampelt auf mono-22.05 kHz

[37] Aufteilung nach Sender bei Generierung aufgelöst.

3.3 Pipeline

Nachdem wir die Grundlagen verstanden hatten und endlich eine solide Datenlage in Aussicht hatten, entwarfen wir die erste Version unserer Pipeline: Abb.3, S.22. Die Schnittstelle zum gesamten Prozess sollte das Tool *Hathor* bilden (3.3.4). (1) Über Hathor sollte sich die Datensammlung mit dem streamripper ansteuern lassen, und über die Verwendung von Last.FM und den dort von der Community bereitgestellten Genre-Tags, hätten selbst genre-unspezifische Sender bezogen werden können. (2) Auch der nächste Schritt – die Wandlung der gerippten MP3-Datein in Spektrogramme – sollte über Hathor gesteuert, und (3) über *transform.py* (3.3.2) durchgeführt werden sollen. (4&5) Das Training und die Verwendung des – nicht entstandenen – Modells (3.3.3) wäre so ebenfalls zentral verwaltet worden.

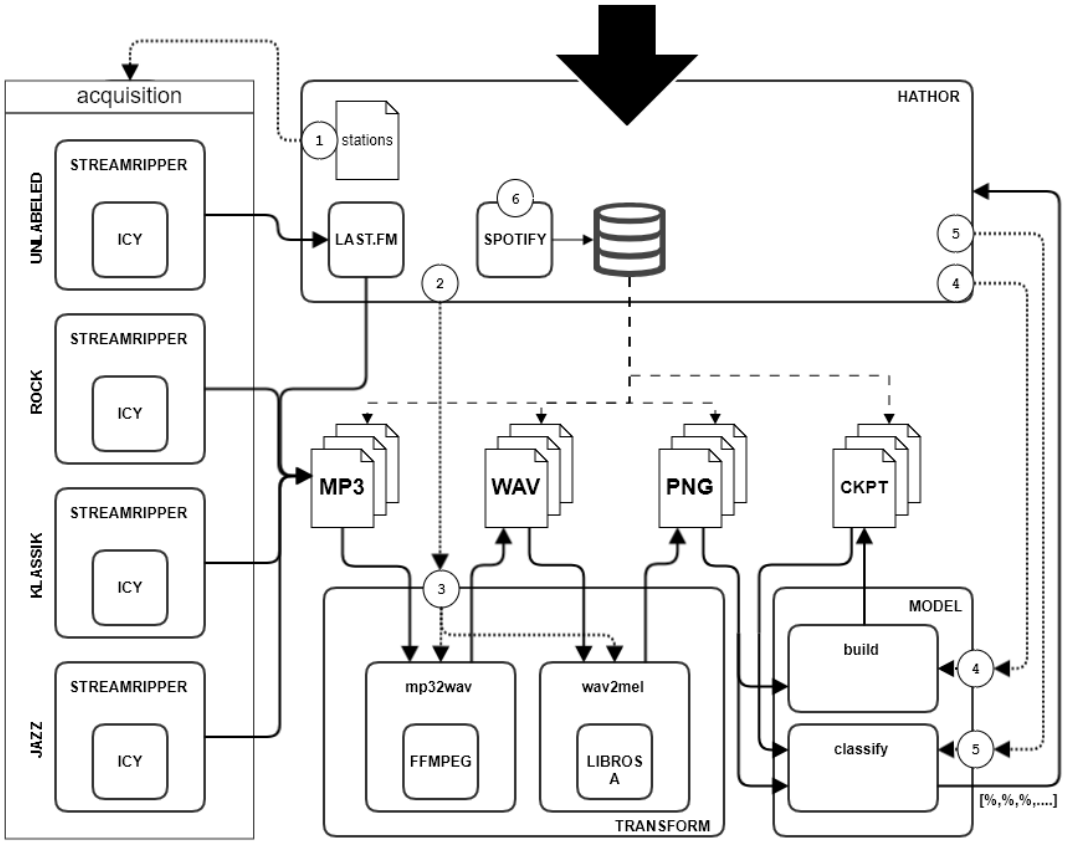


Abbildung 3: Ursprünglich geplante Pipeline

Aufgrund von knapp werdenden Zeit sahen wir uns gezwungen unsere Pipeline zunächst zu fokussieren (Auslassen der Integration mit Hathor)[38] um im Endef-

[38] Siehe Abb.18, S.37

fekt auch diese Version nicht vollständig umsetzen zu können und bei der letztlichen Version landeten: Abb.4, S.23.

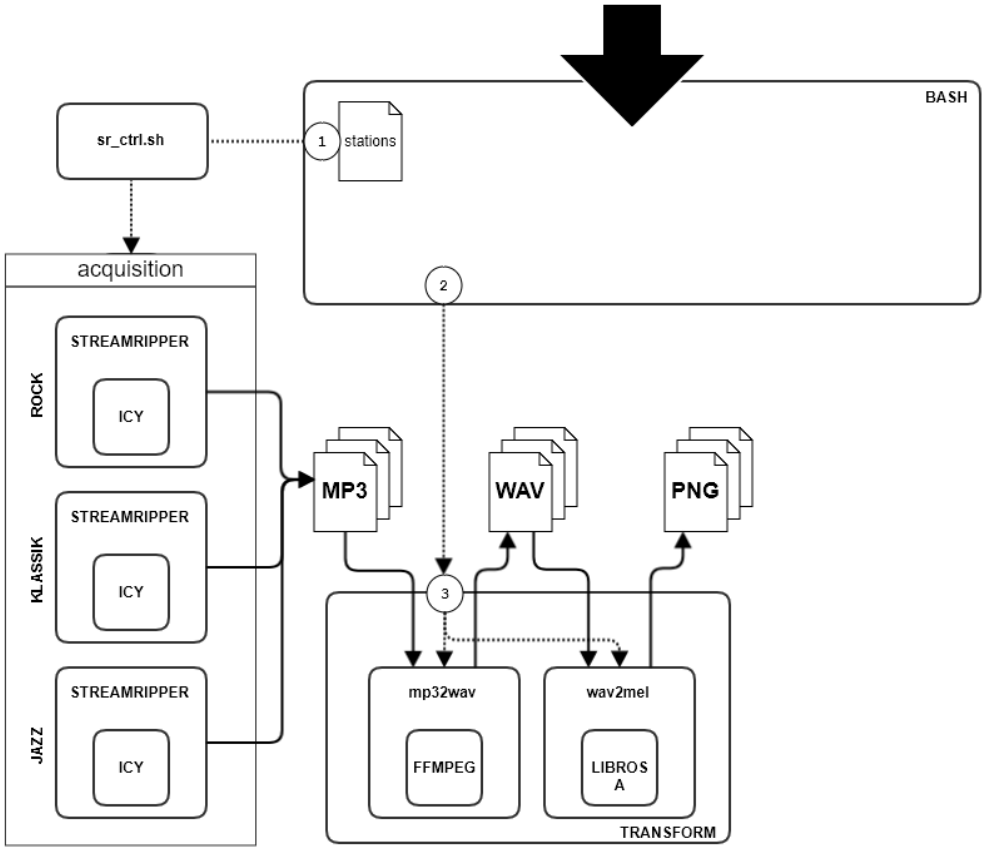


Abbildung 4: Pipeline, Endstand

3.3.1 Streamripper

Als Hotfix für das Wegfallen von hathor zur Steuerung des streamrippers bastelten wir ein kleines Shellscript (sr_ctrl.sh), welches eine Steuerungsdatei (sr_ctrl_stations.txt) mit den entsprechenden Informationen zu Station und Genre einlas und auf dieser Basis die gelisteten Sender aufzeichnete.

3.3.2 mp32wav.py & wav2mel.py & transform.py

Das Programm mp32wav.py fungiert als Python-Wrapper um ffmpeg und wird, wie der Name verrät, zur Konvertierung von den vom streamripper gelieferten mp3 zu wav (zugängliches PCM) verwendet. wav2mel.py wandelt die PCM Daten in MEL-Skalierte Spektrogramme, um einen möglichst simplen und zugleich der Lage nach

der Transduktion entsprechenden Input für unser Netz zu erhalten. transform.py verbindet diese beiden Teile unter einer Schnittstelle, die es erlaubt die Generierung der Spektogramme zu konfigurieren, und für alle Datein in einer angegebenen Verzeichnisstruktur anzuwenden. Die Funktionsweise der einzelenen Programme ist in den entsprechenden Dateien dokumentiert.

Zur Konfiguration gehören dabei unter anderem die Dauer der einzelnen Spektrogramme, die Entscheidung, ob diese mit Überlappung generiert, bunt oder in graustufen gerendert und mit welcher Samplerate die PCM Daten gelesen werden sollen.

3.3.3 model.py

Das letztliche Implementationsziel (abgesehen von der automatischen Datenbeschaffung und -wandlung) haben wir leider nicht erreicht. Geplant war es hier, mit Hilfe von Tensorflow ein Convolutional Neural Network zu implementieren. Die Generierung sämtlicher Inputdaten mit Genre-Label ist zwar abgeschlossen[(39)], und diverse Tutorials zur einarbeitung in das Framework wurden durchlaufen, für die Umsetzung oder wenigstens die Behandlung von Tensorflow in diesem Bericht, haben wir leider keine Zeit mehr gefunden. An der Struktur der Spektrogramme lässt sich jedoch erahnen, dass es unser Plan war, unabhängig von der länge der Lieder, immer gleich große Stücke als Input zu verwenden, und im zweifel sogar zu verschränken (siehe 3.3.2). Hätten wir eine erste Implementation geschafft, so wär das nächste Ziel gewesen, die zeitlichen Zusammenhänge, nicht nur implizit über die Spektrogramme, sondern explizit über Rekurenzen im Netz zu modellieren.

3.3.4 hathor.py & Last.fm

Wir haben uns überlegt, dass es am Besten ist, wenn wir uns ein Tool schreiben, welches den Datensatz verwaltet, sodass wir so wenig wie möglich manuell machen müssen und so potenzielle Fehlerquellen bei unsauberer Manipulation vermeiden können. Als Inspiration haben wir dieses Tool *Hathor* genannt in Anlehnung an die ägyptische Göttin der Musik.

Zunächst soll Hathor Funktionen des Datacleanings erfüllen. Beim Rippen über Streamripper werden immer wieder kurze Jingles und Anmoderationen vor Musiktiteln aufgezeichnet. Diese liegen als eigene .MP3 Dateien vor und würde bei weiterer Verwendung als Musiktitel des Genres ausgewertet werden. Über Hathor sollen sich

[(39)] Der Datensatz wird für einige Wochen nach der Abgabe dieser Arbeit garantiert unter omnesia.org/bdp zu finden sein.

über Filter wie Mindesttitellänge und Mindestbitrate aus dem Datensatz entfernen lassen.

Bei Vorträgen zu Big Data und Data Mining hört man oft, dass jede Art von Rohdaten essentiellen Wert für Analysen bekommen kann. Wir haben versucht diesem Prinzip, in Abwägung mit dem uns zur Verfügung stehenden Speicherplatz, so gut es geht Folge zu leisten. So wird in Hathor eine interne Datenbank aufgebaut, in der versucht wird so viele Metadaten wie möglich zu den Titeln zu sammeln. Neben den Informationen, die man aus den reinen Dateien selber gewinnen kann wie Größe, Länge, Bitrate, Samplerate und Kanäle und den Metadaten aus dem Ripp-Vorgang wie Genre und Station versuchten wir auch externe Datenquellen einzubeziehen. Konkret fragt Hathor über die Spotify Web Api [29] und die LastFM [30] Informationen zu den Musiktitel ab.

Spotify findet bei einem Match des Titels und Interpreten grundlegende Metadaten wie Veröffentlichungsjahr und Album zurück. Über den API Endpunkt *"Audio Features"* gibt Spotify auch Resultate interner Analysen frei, welche Spotify selber verwendet, um seinen Nutzern ähnliche Musik vorzuschlagen. Diese Merkmale sind teilweise musiktheoretische Begriffe wie Tempo(BPM), Lautstärke(dB), Dur/Moll, Taktart oder Tonhöhe. Andere sind weiche Charakteristika wie Acousticness, Tanzbarkeit, Energie(Intensität), Instrumentalität, Speechness oder Stimmung.

Last FM ist ein Musikdienst, der sich auf Musikempfehlung spezialisiert hat und seiner Community von Nutzern ermöglicht, Musik zu mit freien Labeln zu taggen. Über diese Tags erkennt LastFM dann Ähnlichkeit von Musiktiteln und schlägt diese ggf. vor. Diese von den Nutzern annotierten Tags lassen sich sortiert nach der Häufigkeit über eine API zu einem Titel abfragen. In Hathor speichern wir die drei am häufigsten vorkommenden Tags zu einem Musiktitel.

Auch wenn wir diese Metadaten für unser geplante Trainingsmethode über Spektrogramme nicht brauchen glauben wir, dass sie uns trotzdem vom großem Nutzen sein können. Sie geben uns die Freiheit und Agilität gegebenenfalls unseren Ansatz zu verändern und Methoden zu kombinieren, welche vielleicht diese Metadaten verwenden und berücksichtigen. Wir glauben auch, dass diese Metadaten ein gutes Mittel für die Evaluation unseres Klassifikators sein können wenn wir diese Metadaten für das Training des Klassifikators nicht verwendet haben. Nach dem Training wird der Klassifikator nämlich in gewisser Weise eine Blackbox sein, die zwar Fakten wie die Genauigkeit der Klassifikator auf einem Testdatensatz liefert, aber keinen Einblick in den Prozess des Klassifizierens selber geben wird. Hier können wir, unter Einbeziehung der zusätzlichen Metadaten, eine tiefer gehende Evaluation betreiben und gege-

benenfalls Aussagen treffen wie *"Unser Klassifikator funktioniert besonders schlecht bei Rockmusik, die unter XY dB liegt"* oder *"Aus den erfolgreichen Klassifikationen können wir schließen, dass unser Klassifikator Klassik als ein Genre mit schlechter Tanzbarkeit und einer großen Instrumentalität erachtet"*.

artist	track	station	genre	filesize	length	channels	bitrate	samplerate
The XX	Intro	abc	pop	3.4 MB	128	2	320000	44100

album	id	acounsticness	danceability	energy	instrumentalness	key	liveness
xx	123	0.46	0.607	0.789	0.907	9	0.128

loudness	mode	speechiness	tempo	bpm	valence	genre_tag_1	genre_tag_2	genre_tag_3
-8.87	0	0.0278	100	4	0.16	indie	alternative	electronic

Abbildung 5: Beispielhafter Datapoint aus der Hathor Datenbank

4 Fazit

In dieser Projektarbeit konnten wir beide sehr viel dazu lernen. Für uns beide war es das erste große Pythonprojekt und der erste praktische Kontakt mit künstlichen neuronalen Netzen. Da wir uns unser Thema selber aussuchen konnten, haben wir uns für ein Projekt entschieden, welches uns beide, nicht zuletzt durch den zuvor praktisch nicht-existenten Kontakt mit der Materie, fasziniert hat. Diese Begeisterung hat auch während der Arbeitszeit angehalten, und auch in Angesicht immer weiterer Fehlschläge und Misskalkulationen bezüglich der Zeit, bis zum Abschluss nicht nachgelassen.

Es war reizvoll innerhalb der Informatik völlig fremde Bezüge neu verstehen lernen zu können. Dabei konnten wir zusätzlich unsere Recherchefähigkeit verbessern und unser Verständnis für die Tragweite der Informatik erweitern.

Der Prozess der Datenbeschaffung hat sich als mühselig und langwierig herausgestellt. Dies hat aber dazu geführt, dass wir uns schon frühzeitig Gedanken um die benötigte Beschaffenheit des Datensatzes Gedanken gemacht haben und mit diesem hinterher gut arbeiten konnten.

Im Zeitmanagement und der Projektplanung haben wir Fehler gemacht, welche dazu geführt haben, dass wir unser Projekt erst später als erwartet und nicht in dem Umfang, wie zu Anfang gewünscht beendet, konnten. Hier werden wir in Zukunft aus unseren Fehlern lernen und unsere zukünftigen Projekte mit einer zeitlich streng mit Zwischenzielen gegliederten Roadmap durchführen.

Generell hätten wir häufig Zeit sparen können, wenn wir uns ausgiebiger über schon bestehende Lösungen unserer Teilprobleme wie z.B. das Streamripper Projekt informiert hätten, anstatt direkt selber anzufangen eigene Lösungen zu implementieren. Dies ist aber auch der Tatsache geschuldet, dass wir einen starken Anspruch an uns selber hatten und jedes Verfahren, Protokoll und Phänomen welches wir benutzen mussten oder auf welches wir gestoßen sind vollständig verstehen wollten, bevor wir z.B. eine fertige Bibliothek verwendet haben oder eine eigene Implementierung geschrieben haben.

Zusammengefasst sind wir froh das Big Data Projekt gemacht zu haben und können mit Selbstbewusstsein sagen, dass wir vieles in der Projektzeit gelernt haben.

5 Literaturverzeichnis

[1] Kadam, V., Nayak, R. (2016). *Basics of Acoustic Science*. In Padhye, R., Nayak, R. (ed.). *Acoustic Textiles*. Springer, Singapore

[2] `http://asastandards.org/Terms/sound-2/`

[3] Foken, W. (2010). *Einführung in die technische Akustik für Ingeneurstudenten*. Westsächsiche Hochschule Zwickau, University of Applied Sciences.

[4] Möser, M. (2005). *Technische Akkustik*, Springer Verlag Berlin Heidelberg.

[5] Kinsler E., Frey A., Coppens A., Sanders J. (1999). *Fundamentals of Acoustics, 4th Edition*, Wiley.

[6] `https://en.wikipedia.org/wiki/Particle\_velocity`

[7] Behrends J., et al. (2010). *Duale Reihe Physiologie*, Georg Thieme Verlag, Stuttgart, Kapitel 19: *Auditorisches System, Stimme und Sprache*. `https://katalog.ub.uni-heidelberg.de/titel/66761661`

[8] Alberti P. (2006). *The Anatomy and Physiology of the Ear and Hearing*, `https://www.semanticscholar.org/paper/The-Anatomy-and-Physiology-of-the-Ear-and-Hearing-W.Albertia/6a5f0832a948dde736208de5ca02ada86ec6593d`

[9] Fastl H., Zwicker E. (2007). *Psychoacoustics. Facts and Models*, Springer Verlag Berlin Heidelberg.

[10] Sigal E. (2005). *Akkustik. B.5. Ohr und Hören.*, http://www.musig.de/Theorie/Akustik/Akustik06.htm

[11] Donkelaar, H., Kaga, K. (2011). *The Auditory System*, Kapitel 7: Donkelaar et al. (2011). *Clinical Neuroanatomy*. Online: `http://www.springer.com/cda/content/document/cda\_downloaddocument/9783642191336-c1.pdf`

[12] Jensen, K. (2002). *The timbre model*. In: *The Journal of the Acousitcal Society of America 122, 2238 (2002)*.

[13] `https://upload.wikimedia.org/wikipedia/commons/b/bf/Pcm.svg`

[14] Bosi, M., Goldberg, R. E. (2003). *Introduction to Digital Audio Coding and Standards*, The Kluwer international Series in Engineering and Computer Science SECS 721, Kluwer Academic Publishers.

[15] Shannon, C. E. (1948) *A Mathematical Theory of Communication*, in: *The Bell Systems Technical Journal*, Vol. 27, pp379-423, 1984. Online: `http://math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf`

[16] Ampetronic (2013), *Music audio sample of clipping caused by poor amplifier quality or selection*, Video: `https://www.youtube.com/watch?v=SXptusF7Puo`, Letzter Zugriff: 2018-05-05

[17] TheBx2 (2013), *Bit Depth Comparison*, Video: `https://www.youtube.com/watch?v=IPGU\_Wv9VgQ`, Letzter Zugriff: 2018-05-05

[18] Libraray of Congress, *WAVE Audio File Format*. Online: `https://www.loc.gov/preservation/digital/formats/fdd/fdd000001.shtml`

[19] Libraray of Congress, *MP3 File Format*. Online: `https://www.loc.gov/preservation/digital/formats/fdd/fdd000105.shtml`

[20] Kantardzic, M. (2011). *Data Mining: Concepts, Models, Methods, and Algorithms (2nd Edition)*. IEEE Press.

[21] Tchircoff, A. (2017). *The mostly complete chart of Neural Networks, explained*. Online: `https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464`

[22] O'Shea, K., Nash, R. (2015). *An Introduction to Convolutional Neural Networks*. ArXiv e-prints

[23] Free Music Archive: `http://freemusicarchive.org/`, Letzter Zugriff: 2018-05-06

[24] MusicBrainz API: `https://musicbrainz.org/doc/Development/XML_Web_Service/Version_2`, Letzter Zugriff: 2018-05-06

[25] Musixmatch lyrics API: `https://developer.musixmatch.com/documentation`, Letzter Zugriff: 2018-05-06

[26] Spotify companyinfo: `https://newsroom.spotify.com/companyinfo/`, Letzter Zugriff: 2018-05-06

[27] Spotify Android SDK: https://beta.developer.spotify.com/documentation/android-sdk/, Letzter Zugriff: 2018-05-06

[28] Urheberrechtsgesetz § *53 Abs. 1 S. 1*

[29] Spotify Web API: https://beta.developer.spotify.com/documentation/web-api/, Letzter Zugriff: 2018-03-17

[30] Last FM API: https://www.last.fm/de/api, Letzter Zugriff: 2018-03-17

6 Abbildungen

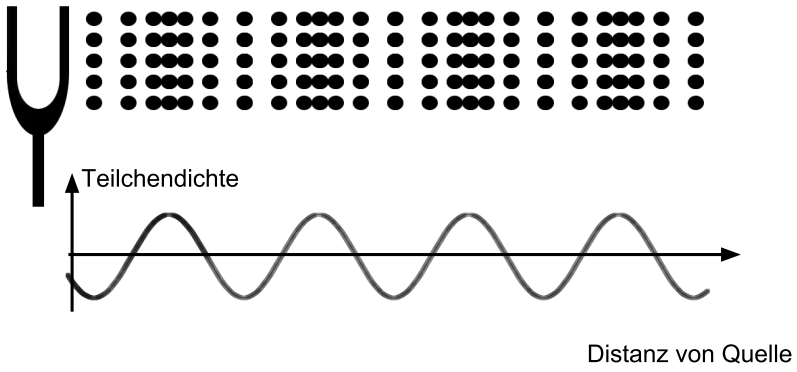


Abbildung 6: Longitudinal Welle; Selbst erstellt

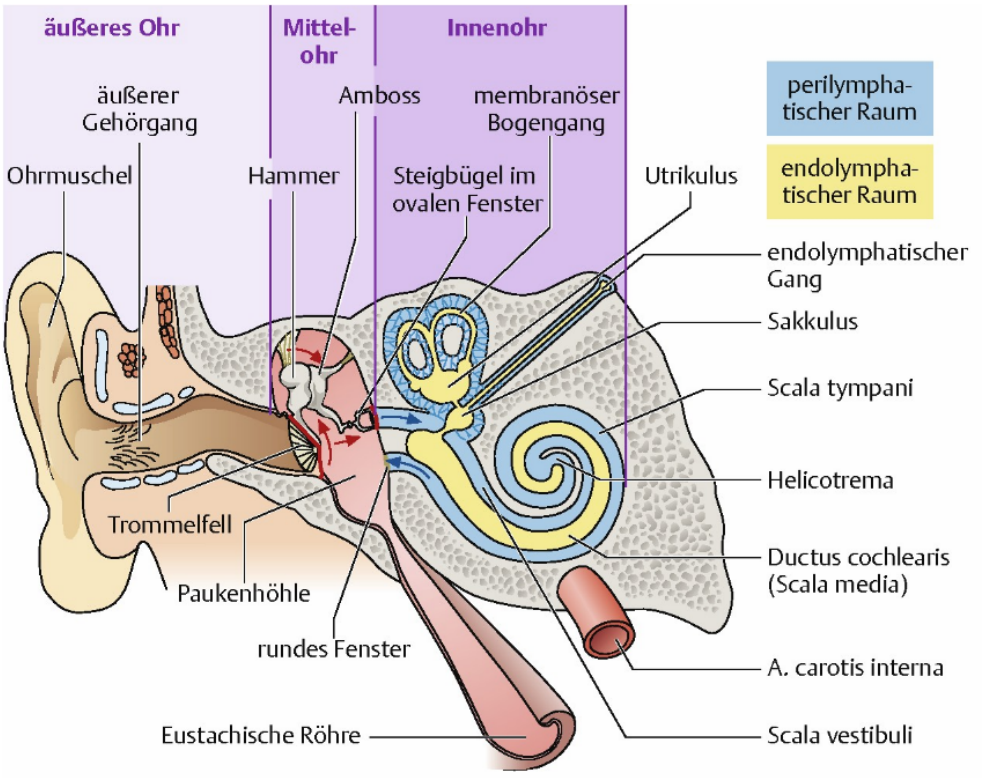
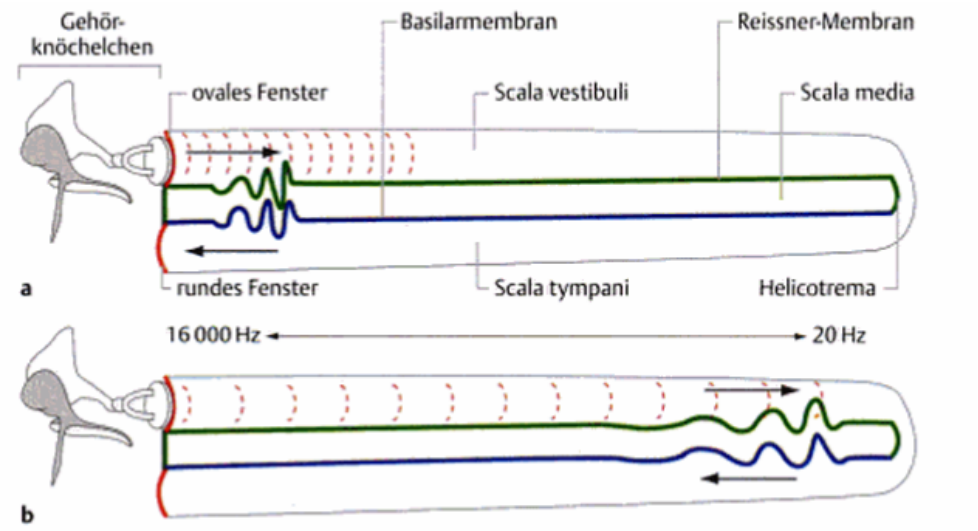


Abbildung 7: Menschliches Ohr [7, S.680]



Schematische Darstellung einer „entrollten" Kochlea: Schallwellen hoher Frequenz erzeugen ein Schwingungsmaximum nahe am Mittelohr **(a)**, langwelliger Schall nahe dem Helicotrema **(b)**.

Abbildung 8: Entrollte Cochlea [7, S.683, Abb.19.6]

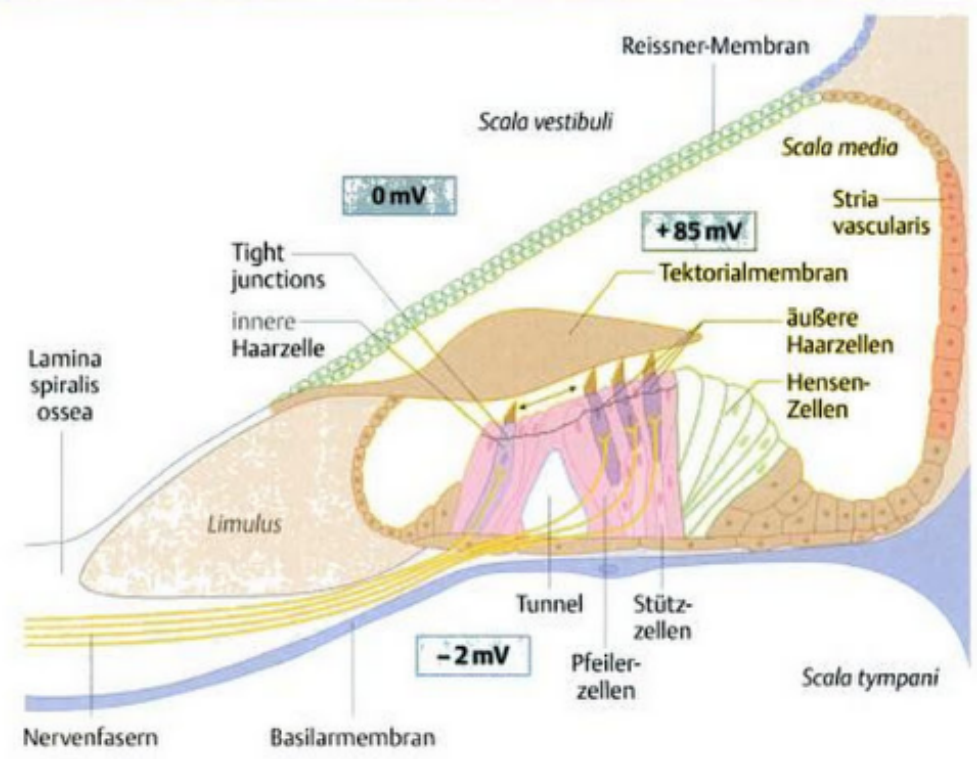


Abbildung 9: Corti Organ und Basilarmembran [7, S.684, Abb.19.7]

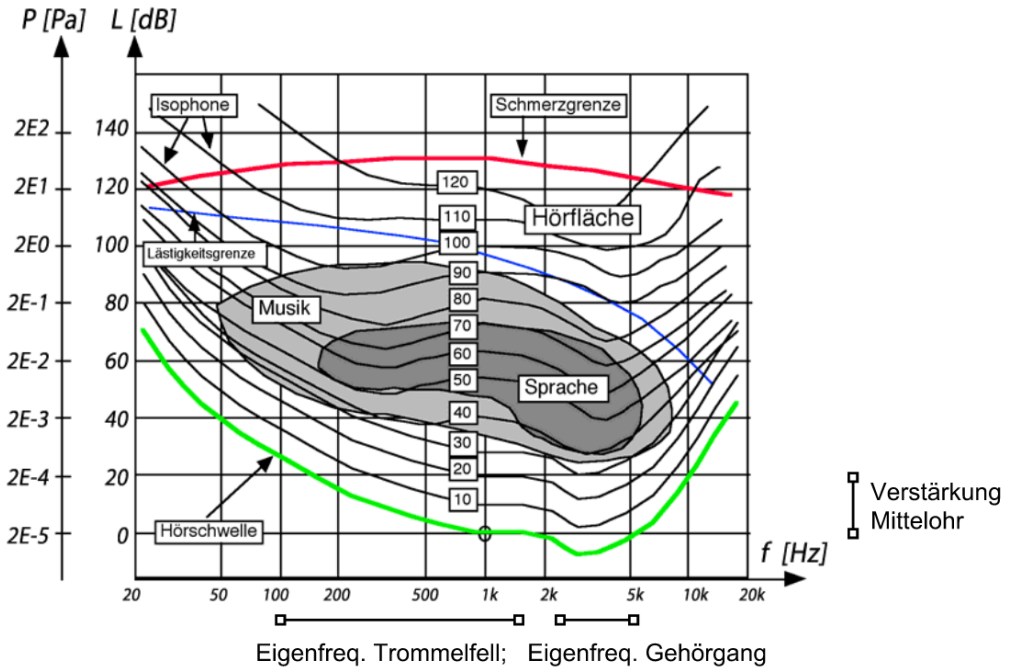


Abbildung 10: Hörbereich; basierend auf [10, Abb. 19-1] und in Anlehnung auf [9, S.17, Fig. 2.1]

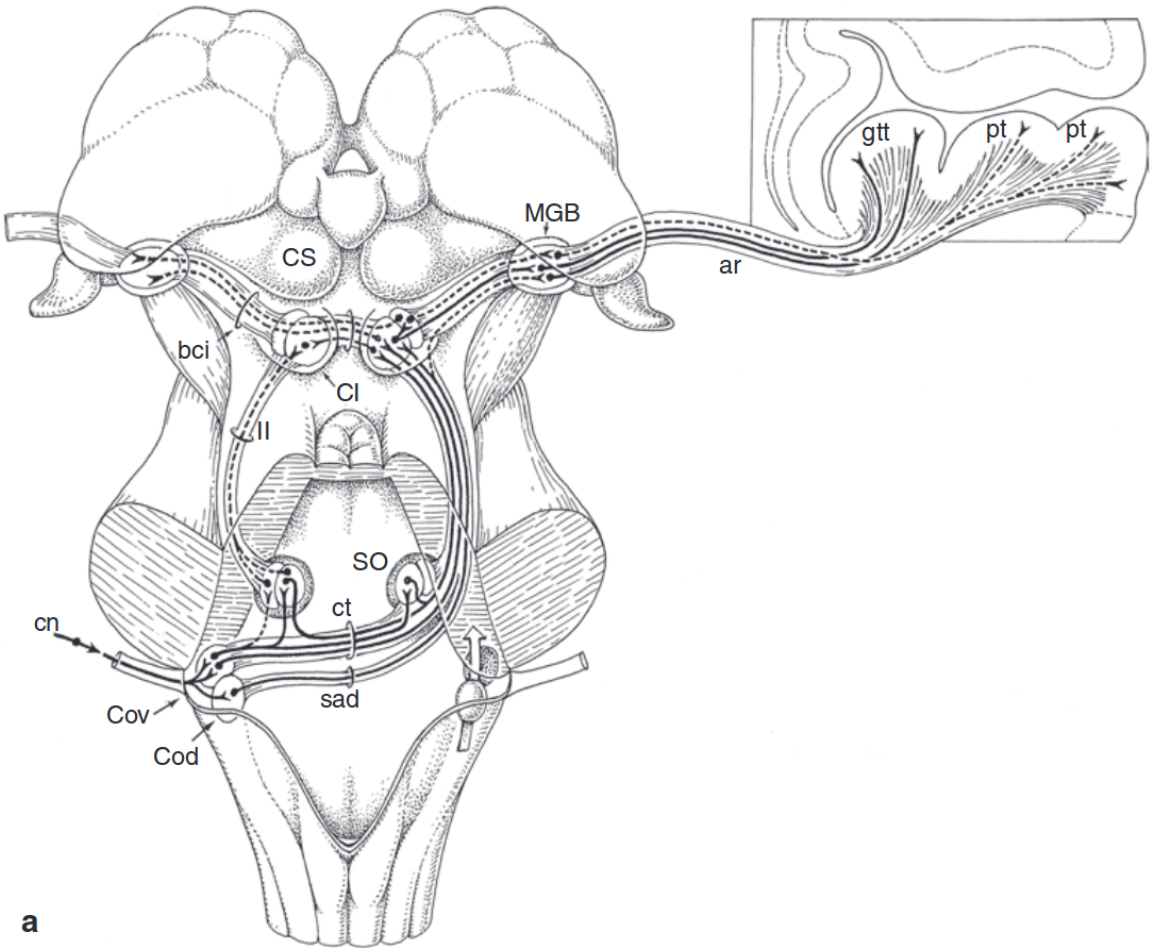


Abbildung 11: Hörbahn: "*ar* accoustic radiation; *bci* brachium of colliculus inferior; *CI* colliculus inferior; *cn* cochlear nerve; *Cod, Cov* dorsal and ventral cochlear nuclei; *CS* colliculus superior; *ct* corpus trapezoideum; *gtt* gyrus temporalis transversus; *ll* lateral lemniscus; *MGB* medial geniculate body; *pt* planum temporale; *sad* stria acoustica dorsalis; *SO* superior olive" [11, S.312, Abb. 7.8 (a)]

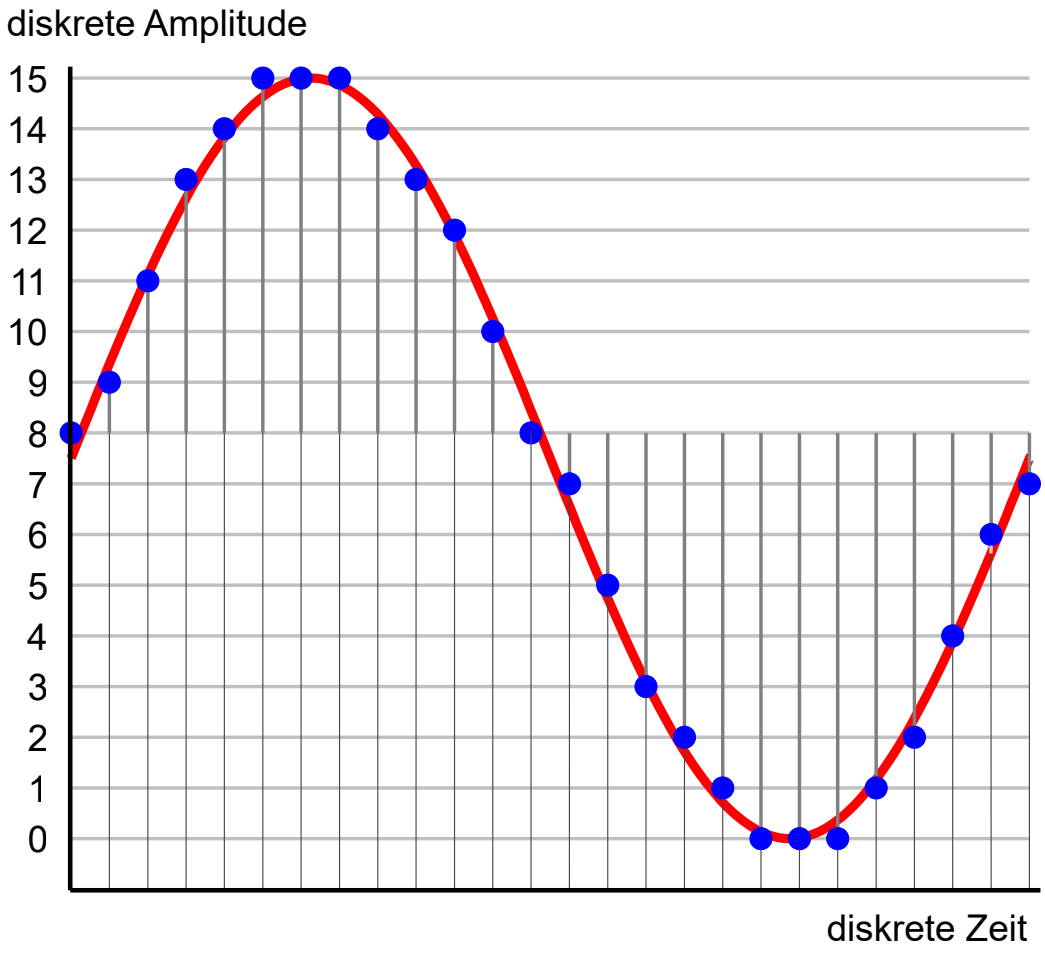


Abbildung 12: PCM Diskretisierung. Basierend auf [13]

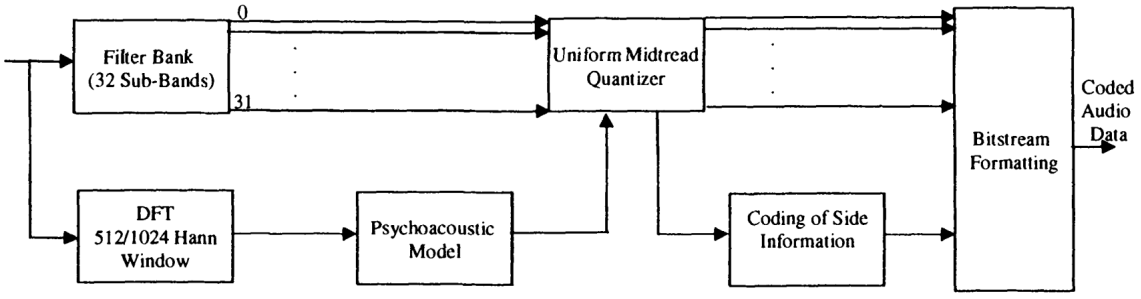


Abbildung 13: MPEG-1 Layer I und II, single channel [14, S.272, 11 - 3.2.1, Fig.3]

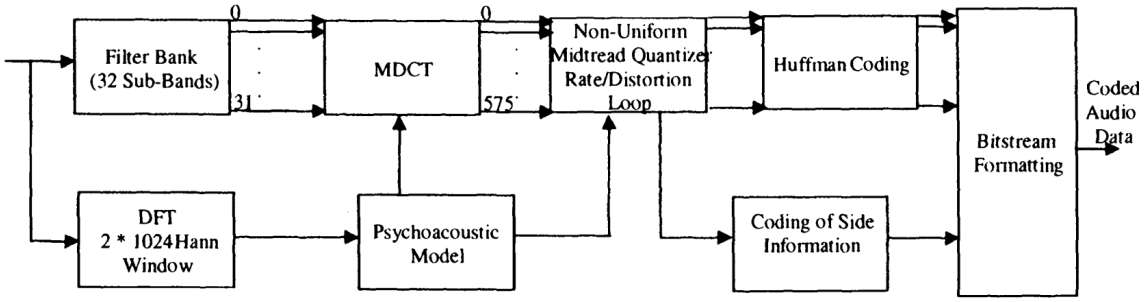


Abbildung 14: MPEG-1 Layer III, single channel [14, S.273, 11 - 3.2.1, Fig.4]

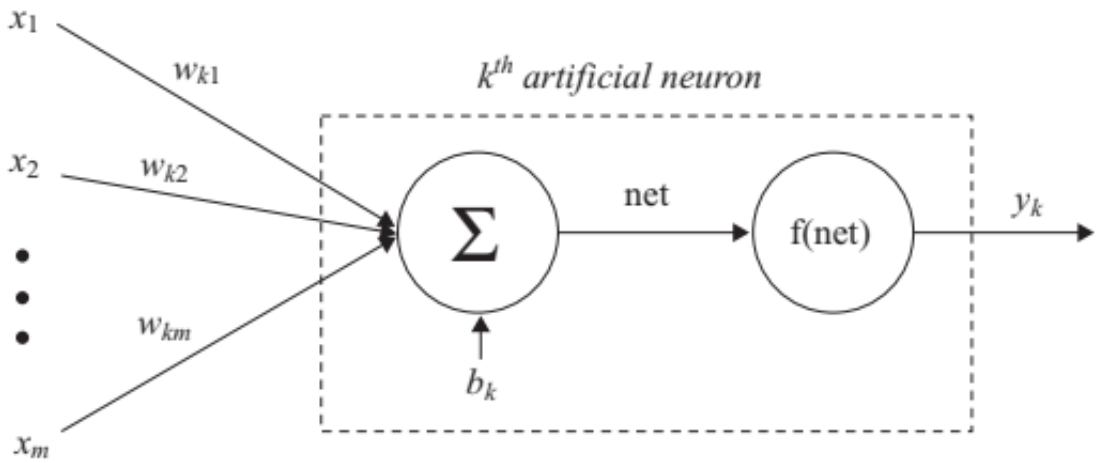


Abbildung 15: Artificial Neuron [20, S.202, Fig 7.1]

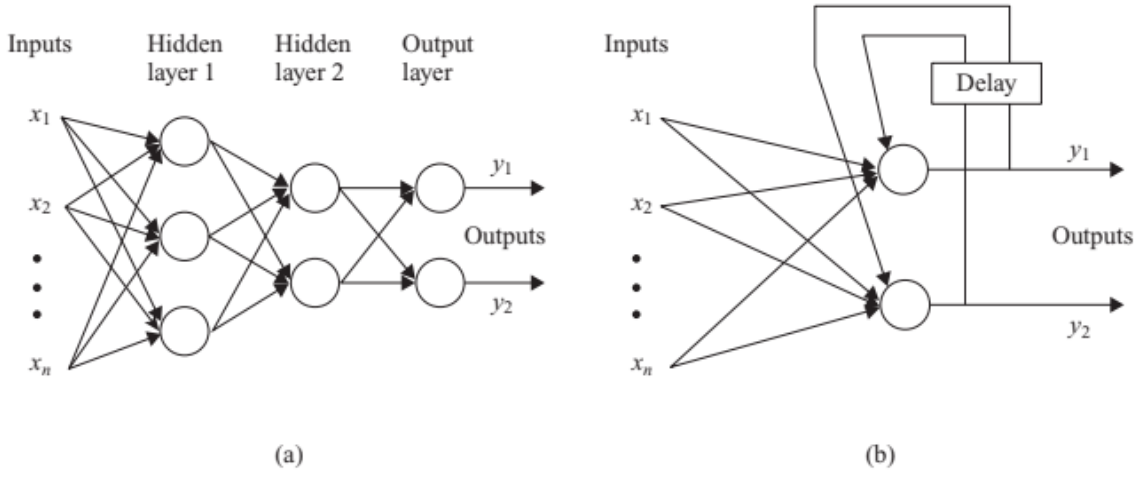


Abbildung 16: (a) Feed Forward Network (b) Recurrent Network [20, S.205, Fig 7.3]

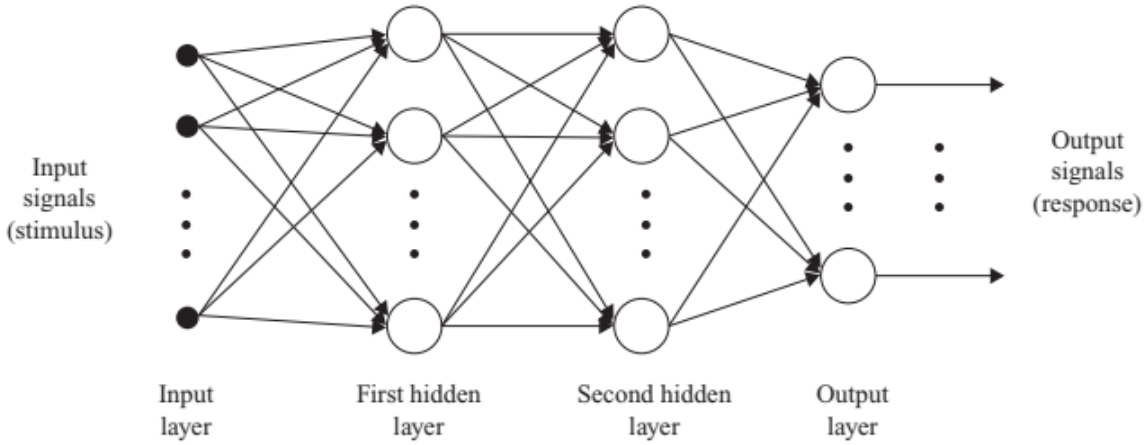


Abbildung 17: Multilayer Perceptron with two hidden Layers [20, S.214, Fig 7.10]

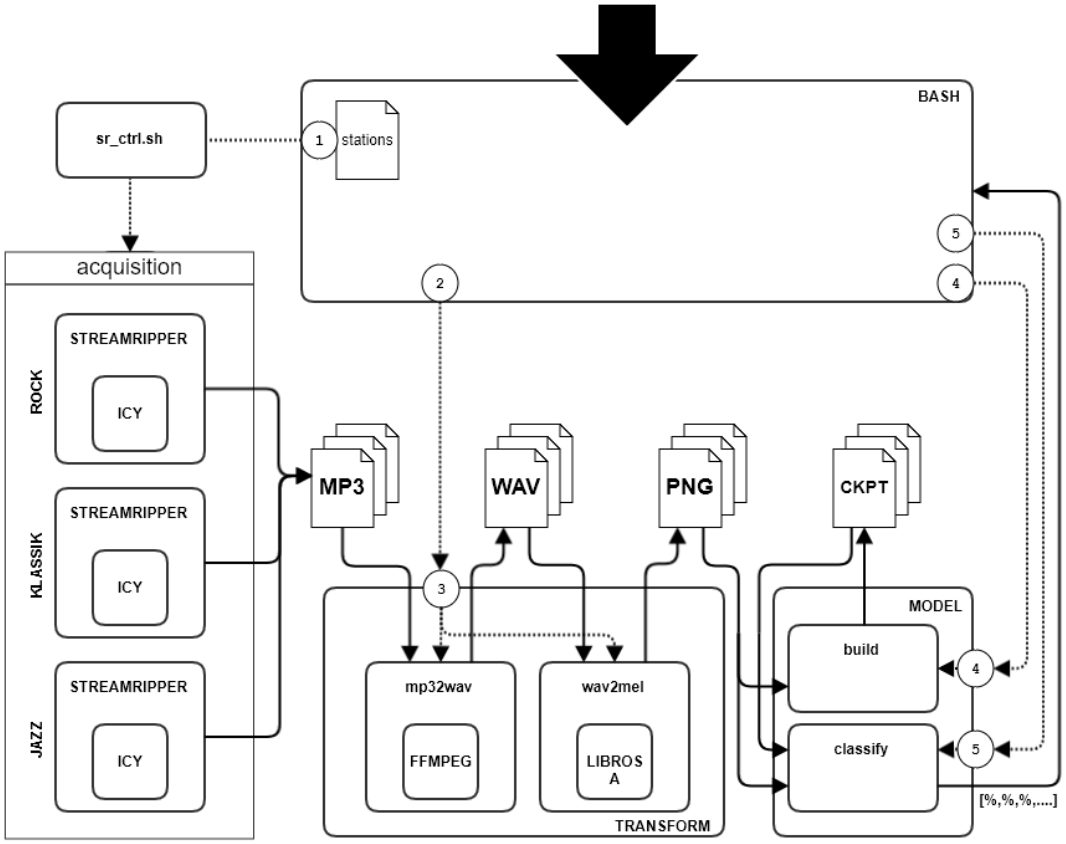


Abbildung 18: Pipeline, Fokusierung