



Hochschule für Technik,
Wirtschaft und Kultur Leipzig

HTWK Leipzig
Fakultät Informatik und Medien

Analyse und Bewertung von Methoden zur Erkennung von Patterns in Musikaufnahmen

Masterarbeit

Zur Erlangung des akademischen Grades Master of
Science (M.SC.)

vorgelegt von:

Nico Mehlhose

Seeburgstraße 60

04103 Leipzig

nico.mehlhose@htwk-leipzig.de

Studiengang:

Informatik

4. Fachsemester

69978

Erstgutachter:

Jean-Alexander Müller

Zweitgutachter:

Martin Grützmüller

Ort und Datum:

Leipzig der 02.06.2020

Inhaltsverzeichnis

| | |
|---|----|
| Danksagung..... | 4 |
| 1. Einleitung..... | 5 |
| 2. Arten von Pattern | 6 |
| 3. Die Fourier-Analyse | 8 |
| 3.1. Grundlagen | 8 |
| 3.2. Die diskrete Fouriertransformation..... | 8 |
| 4. Die Spektralanalyse..... | 11 |
| 5. Convolutional Neural Network | 12 |
| 5.1. Funktionsweise der einzelnen Bestandteile | 13 |
| 5.1.1. Convolutional-Schicht..... | 13 |
| 5.1.2. Pooling-Schicht | 14 |
| 5.1.3. Vollständig Vermaschtes Netzwerk | 15 |
| 5.2. Die ReLU-Aktivierungsfunktionen | 16 |
| 5.3. Backpropagation..... | 17 |
| 5.4. Segmentation Algorithmen..... | 20 |
| 5.4.1. You only look once (YOLO) | 20 |
| 5.4.2. Single Shoot Detector (SSD) | 21 |
| 5.4.3. Faster Region-Based Convolutional Neuronal Network (Faster R-CNN).... | 22 |
| 6. Realisierung der Patternerkennung mittels einem Convolutional Neuronal Network | 24 |
| 6.1. Realisierung des Low-/Band-/High-Pass-Filters | 25 |
| 6.2. Erzeugen der Spektren..... | 28 |
| 6.3. Labeln der Erzeugten Spektren | 28 |
| 6.4. Erstellen des Convolutional Neuronal Networks..... | 29 |
| 7. Patternerkennung mithilfe von Rhythmus und Melodie | 30 |
| 7.1. String basierte Patternsuche | 31 |
| 7.2. Patternerkennung mithilfe von Matrizen | 32 |
| 7.3. Vektorgeometrische Patternsuche..... | 33 |
| 7.3.1. Fünf-Dimensionales Punkt Set | 33 |
| 7.3.2. SIA | 33 |
| 7.4. Umsetzung der Vektorgeometrischen Patternsuche..... | 35 |

| | |
|---|----|
| 8. Patternerkennung mittels SIA und der anschließenden Auswertung durch ein CNN mit YOLOv4..... | 36 |
| 9. Vergleich der unterschiedlichen Verfahren der Patternerkennung | 37 |
| 9.1. Probleme der neuronalen Netze..... | 37 |
| 9.1.1 Patternerkennung mit CNN und Spektrum..... | 37 |
| 9.1.2. Patternerkennung mit SIA und CNN | 38 |
| 9.2. Rechenzeit | 41 |
| 9.3. Anzahl der gefundenen Pattern | 43 |
| 10. Fazit..... | 45 |
| Abbildungsverzeichnis..... | 47 |
| Formelverzeichnis..... | 48 |
| Literaturverzeichnis | 48 |
| Gesprächsleitfaden | 53 |
| Anhang..... | 54 |
| Eidesstattliche Erklärung..... | 56 |

Danksagung

An dieser Stelle möchte ich den Kontrolllesern Maik und Leandra dieser Masterarbeit danken. Des Weiteren ein großes Dankeschön an Raphael Drechsler welcher mit seiner Expertise und seinem Wissen aus so mancher tiefer Note meines Musikwissens eine helle Note hervorgebracht hat.

1. Einleitung

Musikstücke setzen sich bekanntermaßen aus einer Vielzahl von Noten zusammen. Jedoch sind diese in einem Musikstück nicht zufällig angeordnet. Eine bestimmte sich in einem Musikstück wiederholende Anzahl von Noten kann zu Pattern zusammengefasst werden. Die Pattern können dabei markante Melodien eines Songs abbilden.

Wenn Pattern dargestellt werden, können die Information der Struktur, Art, Länge und Position der Patterns nicht nur in der Musikalische Ausbildung, sondern auch für die Identifikation typischer Schemata herangezogen werden.

Diese Abschlussarbeit handelt von verschiedenen Verfahren der Patternextraktion und Darstellung von Pattern eines Musikstücks. Die Verfahren der Patternerkennung gehen dabei über die reine Rhythmuserkennung hinaus. Zur Analyse werden die Patternsuche mittels Stringvergleich, die Patternsuche mittels Matrix, die Vektorgeometrische Patternsuche, die Patternsuche mithilfe Convolutional Neuronal Networks, welches Pattern in den Spektren der Musik erkennen soll und die Convolutional Neuronal Network welches mithilfe eines Zwischenergebnis der Vektorgeometrischen Patternsuche Pattern erkennen soll verwendet.

In dieser Arbeit werden in Kapitel zwei alle Pattern genannt, welche mit den oben genannten Verfahren, erkannt werden sollen. Kapitel drei bis fünf bilden die Heranführung an die Variante des Convolutional Neuronal Networks, welches mit den Spektren der Musikstücke arbeitet. Kapitel drei handelt um die Fourier-Analyse welches für die Erzeugung von Spektren essenziell ist. Kapitel vier dreht sich um die Spektralanalyse, mit welcher die Spektren mittels einer diskreten Fouriertransformation erzeugt werden. Der darauffolgende Abschnitt behandelt die Grundlagen eines Convolutional Neuronal Networks sowie dessen Aufbau. Kapitel sechs baut auf den eben beschriebenen Kapiteln auf, indem es die erklärten Grundlagen für die Umsetzung der Patternerkennung des Convolutional Neuronal Network mit Spektren verwendet.

Der darauffolgende Abschnitt erläutert die Grundlagen und Umsetzung der String basierten Patternsuche, Matrix basierten Patternsuche und der Vektorgeometrischen Patternsuche. Kapitel acht beschreibt die Kernidee der Patternsuche mittels Convolutional Neuronal Network und der Vektorgeometrischen Patternsuche.

Der letzte Teil der Arbeit handelt von der Bewertung der fünf verschiedenen Verfahren. In diesem Teil werden ebenfalls die Ergebnisse der Varianten diskutiert. Folgende Ergebnisse wurden erzielt: Die Variante mit neuronalem Netz und Spektrum ist aufgrund von eingeschränkten Hardwareressourcen nicht trainierbar. Das zweite neuronale Netz mit Vektorgeometrischer Erkennung brachte keine brauchbaren Klassifikationen hervor, woraufhin beschrieben

wurde, in welchem Einsatzgebieten diese Art der Erkennung sinnvoller und erfolgreicher wäre.

Die verbliebenen drei Varianten haben gute bis sehr gute Ergebnisse erzielt. Verglichen wurde dabei die Rechenzeit, Anzahl der gefundenen Pattern und der Anwendungsmöglichkeiten hinsichtlich von Musikgenres. Dabei haben die String basierte Patternsuche und die Matrix basierte Patternsuche gleichermaßen erfolgreich die Tests bestanden. Lediglich die Vektorgeometrische Variante SIA hat sowohl in der Laufzeit als auch in der Anzahl der gefundenen Pattern Schwächen bewiesen.

2. Arten von Pattern

Bevor mit den Kapiteln für die Grundlagen angefangen wird, handelt dieses Kapitel von den zu erkennenden Pattern. Dabei sei angemerkt, dass nicht alle Pattern der Musiktheorie, erkannt werden sollen.

Die zu erkennenden Pattern sind folgende:

- Wiederholung
- Keychange
- vertikale Spiegelung der Noten oder auch melodische Inversion

Wenn diese Pattern erkannt wurden, können diese für weitere Informationen benutzt werden. Weitere Informationen werden jedoch nicht aus den Daten ausgelesen, da das Auslesen weiterer Informationen den zeitlichen Rahmen der Abschlussarbeit übersteigen würden. Somit kann das Auslesen weiterer musikalischer Merkmale Teil einer weiteren Abschlussarbeit darstellen.

Wiederholung

Die Wiederholung der gespielten Noten ist die am einfachsten zu erkennende Struktur in Musikstücken. Die mindestlänge einer Wiederholung wird in dieser Arbeit bei drei Noten liegen, da drei Noten ausreichen, um einen Akkord darzustellen. (Drechsler, 2020), (Kriesel, 2013) Ein Akkord kann auch durch zwei Noten dargestellt werden. Jedoch würde diese Länge bei dem Suchen nach Pattern zu „false positives“ führen und somit die Ergebnisliste unnötig mit irrelevanten Einträgen füllen. Die Maximallänge eines Patterns liegt bei Tracklänge ganzzahlig geteilt durch zwei.

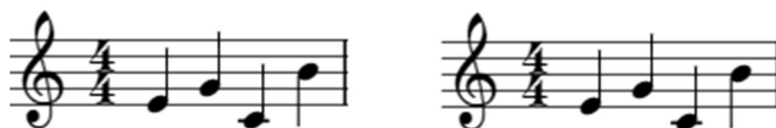


Abbildung 1: Originalpattern (links) mit dem dazugehörigen zu erkennenden Pattern einer Wiederholung (rechts)

Abbildung 1 zeigt ein Beispiel einer Wiederholung, welche in diesem Werk erkannt werden soll.

Keychange

Ein Keychange ist eine Erhöhung/Verringerung der Notenhöhe. Dabei ist es irrelevant wie groß der Betrag des Unterschiedes ist. Es zählt lediglich, wie schon bei der Wiederholung, dass mindestens drei aufeinander folgende Noten denselben Betrag aufweisen. (Drechsler, 2020), (Kriesel, 2013) Die Mindestlänge von drei Noten besteht abermals, da ein Keychange der Länge eins das gesamte Musikstück als Pattern erkennen würde und der Länge zwei ebenfalls zu *false positives* führen würde. Die Maximallänge beträgt, wie auch schon bei der Wiederholung, Tracklänge ganzzahlig geteilt durch zwei.

Untergestellte Abbildung zeigt einige Beispiele für den Keychange.



Abbildung 2: Originalpattern (links) mit einem Keychange um 4 Halbtöne nach oben (mitte) und einer Oktave nach unten (rechts)

Melodische Inversion

Die melodische Inversion ist das Spiegeln der Noten an einer Spiegelachse in horizontaler Richtung. Somit besitzt diese Art des Pattern auch Eigenschaften des Keychanges. Jedoch werden die Noten bei der melodischen Inversion bei zunehmender Tonhöhe im Originalpattern tiefer. Eben beschriebener Fall existiert analog, wenn die Note im Originalpattern tiefer wird. (Drechsler, 2020), (Kriesel, 2013) Die Mindestlänge der Pattern ist dieselbe wie bei dem Keychange und der Wiederholung.



Abbildung 3: Originalpattern (links) mit einer vertikalen melodischen Inversion an Note C5 (mitte) und E4 (rechts)

Sollten mehrere Instrumente auf einen einzelnen MIDI-Track liegen, so können die Pattern nicht separat für jedes Instrument untersucht werden. Wenn mehrere Instrumente auf einem einzelnen MIDI-Track dargestellt sind, werden kann nicht garantiert werden, dass alle Pattern erkannt werden. Einzig die

Patternerkennung mit Convolutional Neuronal Network und Spektrum ist von eben beschriebenen Sachverhalt ausgenommen.

3. Die Fourier-Analyse

3.1. Grundlagen

Für jedes wissenschaftliche Feld, welches mit mechanischen Schwingungen, elektrischen Schwingungen oder auch mit der Bildverarbeitung in Berührung kommt, ist die Fourier-Analyse ein wichtiges Werkzeug. Die Fourier-Analyse beruht auf der Grundaussage von Jean Baptiste Joseph Fouriers Forschung aus dem Jahre 1822, welche besagt, dass jede Schwingung mithilfe von unendlich vielen Sinus- und Kosinus-Schwingungen zusammengesetzt werden kann. (Strick, 2012)

Somit kann auch jede periodische Schwingung wieder in ihre Einzelteile zerlegt werden. Dieser Vorgang wird Fourier-Analyse genannt. Dabei gibt es, je nach Eigenschaft der Funktion, vier verschiedene Arten der Fouriertransformation.

- Fourierreihe
- Kontinuierliche Fouriertransformation
- Diskrete Fouriertransformation (DFT)
- Fouriertransformation für zeitdiskrete Signale (DTFT)

In dieser Arbeit werden lediglich die DFT oder die DTFT betrachtet, da diese die einzigen Transformationen sind, die von einem Computer ausgeführt werden können. Die DFT/DTFT ist für den Computer ausführbar, weil diese diskrete Werte und eine endliche Länge besitzen. (Smith S. W., 1997)

3.2. Die diskrete Fouriertransformation

Wie im vorherigen Abschnitt beschrieben, ist die diskrete Fouriertransformation die einzige Transformation, welche vom Computer berechnet werden kann. Um zu beweisen, wieso die DFT diskret und endlich ist, wird die Formel der DFT der Formel der kontinuierlichen Fouriertransformation (CTFT), welche kontinuierlich und unendlich lang ist, gegenübergestellt und die Unterschiede aufgezeigt.

$$X(F) = \int_{-\infty}^{\infty} x(t) * e^{-i2\pi Ft} dt$$

Formel 1: Formel der CTFT (Weisstein, Discrete Fourier Transform, 2015)

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{i2\pi kn}{N}}$$

Formel 2: Formel der DFT (Weisstein, Discrete Fourier Transform, 2015)

In Formel 2 ist N die Anzahl der Samples und k das k -te Sample Bin. Der Unterschied der beiden Formeln liegt in dem Tausch des Integrals durch die Summe und im Exponenten der e-Funktion. Das Integral in der DFT entfällt und wurde durch Sigma ersetzt, da die DFT mit konkreten Zahlenwerten rechnet und nicht mit Flächen. Die Exponenten unterscheiden sich darin, dass die Frequenz F aus der CTFT mit k/N substituiert wurde. Aus der DFT ist n gleichzusetzen mit dem aus der CTFT stammenden t (Weisstein, 2015). Die DFT-Formel kann vereinfacht werden, indem $\frac{2\pi kn}{N}$ mit b_n und die e-Funktion durch die Eulersche Identität substituiert wird. Dadurch erhält man folgende Formel. (Thormählen, 2018) (Smith J. O., 2007)

$$X_k = x_0(\cos(-b_0) + i\sin(-b_0)) + x_1(\cos(-b_1) + i\sin(-b_1)) + \dots + x_n(\cos(-b_n) + i\sin(-b_n))$$

Formel 3: Umformulierte DFT-Formel

Nachdem für die x -Werte die einzelnen Samplewerte eingetragen werden, muss die Formel berechnet werden. Das Ergebnis aus dieser Formel besteht aus einem Real- und einen Imaginär Anteil. Die Real- und Imaginären-Anteile können dann in ein Koordinatensystem eingetragen werden, wobei der Realteil die x-Achse und der imaginäre Anteil die y-Achse darstellt. Der Winkel des eingetragenen Punktes zur positiven x-Achse beschreibt die Phasenverschiebung der Schwingung und die Entfernung des eingetragenen Punktes zum Koordinatenursprung beschreibt die Amplitude der X_k -ten Schwingung.

$$\text{Amp} = \sqrt{\text{Re}_k^2 + \text{Im}_k^2}$$

Formel 4: Berechnung des Betrages eines Vektors

Welche Frequenz die X_k -te Schwingung aufweist, hängt von der Abtastfrequenz und der Anzahl der Abtastpunkte ab. Die Frequenzdifferenz der Punkte berechnet sich durch *Abtastfrequenz/Abtastpunkte*. (Smith J. O., 2007)

Bevor Formel 4 benutzt werden kann, um die Amplitude zu berechnen, müssen alle Samplewerte, deren Index \geq Anzahl Samples/2 sind, gelöscht werden. Dieser Wert wird *Nyquist-Limit* genannt. Es besagt, dass alle Ergebnisse über diesem Limit gelöscht werden und die Ergebnisse der Samples, die unter dem Nyquist-Limit liegen, verdoppelt werden. (Weisstein, 2005) Zuletzt müssen die jetzigen Resultate durch die Anzahl der Samples N geteilt werden. (Smith J. O., 2007)

Bei der DFT können jedoch zwei Fehler auftreten. Zum einen der *Leakage-Effekt* und zum anderen *Aliasing*. Leakage tritt auf, wenn die Schwingung in dem betrachteten Zeitabschnitt nicht perfekt periodisch ist.

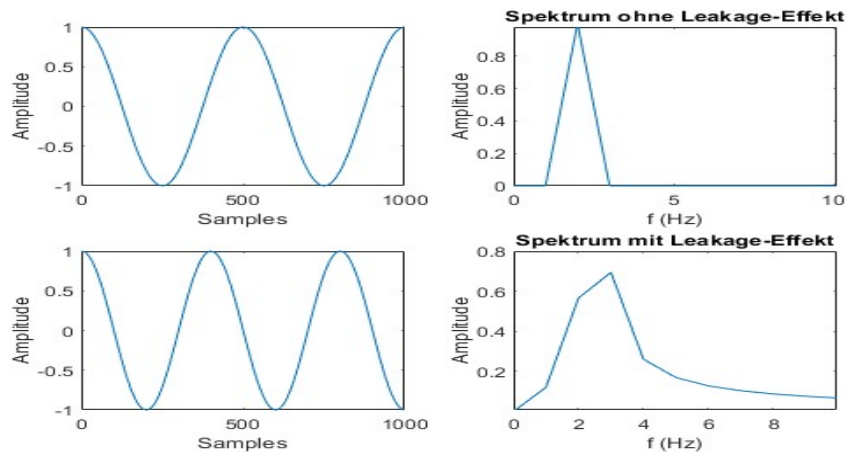


Abbildung 4: Aufzeigen des Leakage-Effektes mit einer Funktion von 2 Hz (oben) und 2.5 Hz (unten)

Wie in der vorangestellten Abbildung zu sehen, ist in dem zweiten Zeitfenster eine Sprungstelle zu sehen, wenn der Zeitausschnitt periodisch weitergeführt werden würde. Die Sprungstelle befindet sich an den Rändern des Ursprungssignals. Durch den daraus resultierenden ungleichmäßigen Kurvenverlauf kommt es zu einer „Verschmierung“ des Spektrums. Der *Leakage*-Effekt kann mit Fensterfunktionen abgeschwächt werden. Wie in Abbildung 5 zu sehen ist, schwächen diese Funktionen die Ränder des Ursprungssignals so ab, dass diese gegen Null gehen. Dadurch treten beim Wiederholen des Signals nur noch geringe Sprünge im Kurvenverlauf auf (Roberts, 2017). Einige Fensterfunktionen sind unter anderem Barlett-, Gauß- oder die Hanning-Fensterfunktion. (Weisstein, 2015)

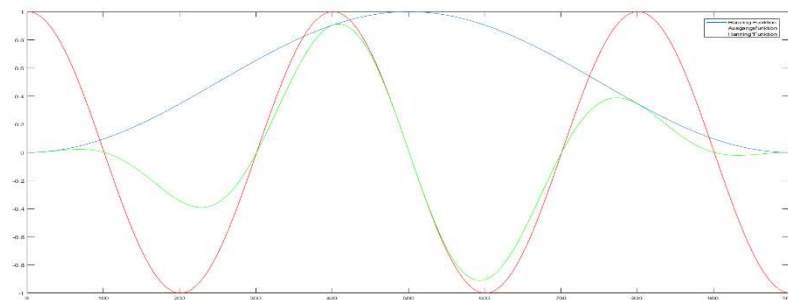


Abbildung 5: Anwenden der Hanning-Fensterfunktion auf eine Schwingung von 2.5 Hz. Die rote Linie stellt die Ausgangsfunktion dar. Blau stellt die Hanning-Funktion dar und die grüne Linie ist die berechnete Funktion.

Aliasing tritt auf, wenn die Samplerate zu gering ist. Dadurch können keine hohen Frequenzen erkannt werden, da für die Berechnung von hohen Frequenzen nicht genug Zwischenpunkte existieren. Um Aliasing zu verhindern, muss die Samplerate angehoben oder das Signal vorgefiltert werden, um die zu hohen Frequenzteile zu minimieren. (Roberts, 2017)

4. Die Spektralanalyse

Die Spektralanalyse eines Musikstücks besteht aus vielen einzelnen Fouriertransformationen, die über den gesamten Verlauf des Stücks durchgeführt werden. Sie gibt uns die Möglichkeit, ein Musikstück als Bild darzustellen und eine Bildanalyse in einem Convolutional Neuronal Network durchzuführen. Dabei hängt die Länge der einzelnen Fouriertransformationen und die Auflösung der Frequenzen von der Größe des zu betrachtenden Fensters ab. Bei einer Samplerate von 44100 Samples pro Sekunde, was als eine Standardgröße bei Musik angesehen werden kann (Teufel, 2019), bedeutet dies, dass der Abstand zwischen 2 Samples rund $22,67 \mu\text{s}$ beträgt. Bei einer Fenstergröße von 32 Samples steht somit die zeitliche Genauigkeit im Fokus. Ein Fenster ist im obigen Beispiel 0,7 ms lang. Jedoch wird dabei die Frequenz vernachlässigt. Dies beruht auf dem in Punkt 3.2. beschriebenen Nyquist-Limit, wodurch der Frequenzbereich bei 32 Samples in 15 gleichgroße Bereiche geteilt werden muss. Bei 44100 Samples pro Sekunde und der daraus resultierenden maximalen Frequenz von 22050 Hz besitzt jeder Bereich ein Frequenzdelta von 1470, was für eine Patternerkennung unbrauchbar ist, wie Abbildung 6 a aufzeigt.

Ist jedoch das Fenster groß, gibt es eine feine Unterteilung der Frequenz aber eine ungenaue zeitliche Unterteilung. Analog zu dem obigen Rechenbeispiel aber mit einer Fenstergröße von 32768 Bins bedeutet dies, dass das Fenster circa 0,74 Sekunden groß ist und jeder Frequenzbereich ein Delta von, rein rechnerisch, 1,345 Hz hat. Um die großen Zeitintervalle auszugleichen, können Bereiche auch überlappt werden. Dadurch wird das Spektralbild, je nach Überlappungsgrad, genauer. (Cannam, Landone, & Sandler, A Brief Reference, 2010)

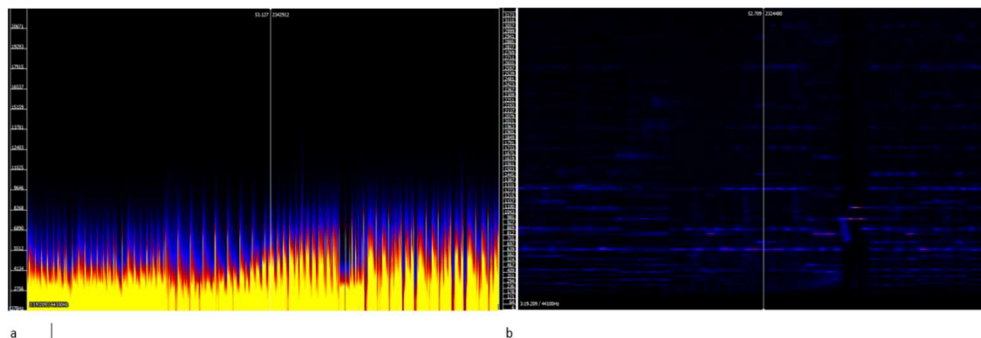


Abbildung 6: Spektrum des Liedes NIVIRIO - Flashes

a) Spektrum mit 32 Bins berechnet

b) Spektrum mit 32768 Bins berechnet (herangezoomt)

5. Convolutional Neural Network

Um die Spektren, welche vergleichbar mit Abbildung 6.b sind, automatisch verarbeiten zu können, wird ein neuronales Netzwerk verwendet. Dafür wird ein Convolutional Neural Network (CNN) benutzt. Diese sind, gegenüber anderer Arten von neuronalen Netzen durch ihren Aufbau schneller zu trainieren und genauer in der Klassifikation. Die Genauigkeit der Klassifikation ist dennoch der wichtigste Vorteil, weshalb ein CNN für die Patternerkennung verwendet wird. In diesem Gebiet sind CNNs seit circa 9 Jahren anderen Netztypen überlegen. (Imagenet, 2017)

Der zweite Vorteil gegenüber anderen neuronalen Netzwerken ist die Effizienz. Durch die Pooling-Schichten der CNNs wird der Rechenaufwand, in Abhängigkeit von der Größe der Poolingmatrix, verringert. Die bekanntesten Pooling Verfahren sind das *max-Pooling* und das *average-Pooling*. Hierbei wird nur ein Wert aus einer vordefinierten $m \times n$ Poolingmatrix übernommen. (Karpathy, 2018)

CNNs haben einen unterschiedlichen Grundaufbau gegenüber herkömmlichen Arten von neuronalen Netzen (NN). NN besitzen mindestens 2 Schichten. Eine eindimensionale Eingabe-Schicht und eine eindimensionale Ausgabe-Schicht. Optional können versteckte Schichten eingefügt werden und vollständig oder teilweise mit der Nachfolgerschicht verbunden werden. (Beck & Rey, 2018) Durch die Eindimensionalität der Input-Schicht können Daten nur als Vektor an das NN weitergegeben werden. Des Weiteren werden die Input-Schicht und die erste Hidden Layer meist vollständig miteinander verbunden. Bei RGB-Bildern wird jeder Farbkanal einzeln gespeichert, was zu einer Dreidimensionalität führt. Somit ist ein 30x30 Pixel großes Bild nicht nur 900 Pixel groß, sondern besitzt eine Größe von 2700 Pixel. Dies führt dazu, dass bei einer Vollvermaschung mit der ersten Hidden Layer 7,29 Mio Gewichte angelern werden müssen. Dadurch ist die Verarbeitung von Bildern nur durch einen hohen Rechenaufwand möglich. CNNs haben den eben beschriebenen Aufbau und Nachteil nicht. Sie besitzen eine bis zu dreidimensionale Input-Schicht, womit RGB-Farbbilder effizient an das Netz gegeben werden können. Außerdem wird die erste Hidden Layer nicht vollständig mit der Input-Schicht verbunden. Die Verbindungen geschehen nur in einer räumlich abgegrenzten Region. Wie groß die Region ist, hängt von der Größe des *Convolutional Kernels* oder des *Pooling Kernels* ab. (Karpathy, 2018) Bei einem 3x3 Convolutional Kernel wären dies, bei obigem Beispiel, lediglich 24843 anzulernende Gewichte. Somit muss das herkömmliche NN zwischen Input-Schicht und erster Hidden Layer circa 290 mal mehr Gewichte anlernen als das CNN.

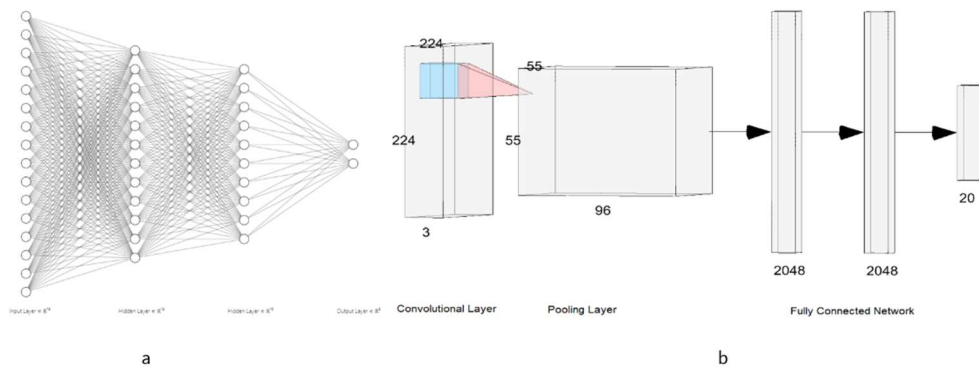


Abbildung 7: a) vollständig vermaschtes Netzwerk
 b) Convolutional Neural Network mit einer Convolutional-Schicht einer Pooling Schicht und einem vollständig vermaschten Netzwerk

Quelle: Abbildungen wurden mittels folgenden Online-Tool erstellt:
<http://alexlenail.me/NN-SVG/index.html>

Die zweite Komponente der CNNs ist die *Convolutional*-Schicht, welche Matrizen über die Pixelmatrix der Vorgängerschicht schiebt, um gewisse Eigenschaften stärker hervor zu heben. Die nächste Komponente ist die *Pooling*-Schicht. Das Pooling bewirkt, dass gewisse Informationen, innerhalb einer $m \times n$ Matrix, verworfen werden. Dadurch wird die Anzahl der Pixel des Bildes verkleinert. Der letzte Baustein ist ein vollvermaschtes neuronales Netz. Das vollvermaschte Netz wird benötigt um die verarbeiteten und verkleinerten Bilder, welche in Form von Feature Maps vorliegen, zu kategorisieren. (Math Works)

Die eben beschriebenen Bausteine müssen nicht in der vorgestellten Reihenfolge vorkommen. Es können mehrere Convolutional-Schichten hintereinander realisiert werden, bevor eine oder mehrere Pooling-Schichten implementiert werden. Diese Strukturen können sich dabei in ein und demselben Netz auch wiederholen. Die einzigen Strukturen, welche einen festen Platz in der Topologie des CNN besitzen, ist die Input-Schicht, welche am Anfang implementiert sein muss, und das vollvermaschte neuronale Netz mit der dazugehörigen Output-Schicht, welches das letzte Segment des CNN's darstellt.

5.1. Funktionsweise der einzelnen Bestandteile

5.1.1. Convolutional-Schicht

Die Convolutional Schicht ist der Hauptteil eines CNN. Bei der in dieser Schicht stattfindende Faltung (engl. convolution), werden vom Ersteller gewünschte oder vom Netz vordefinierte Merkmale extrahiert. Gewünschte Merkmale können dabei Kanten in horizontaler, vertikaler oder diagonaler Richtung darstellen. Weitere Merkmale sind jedoch auch möglich. Die Merkmale werden mit Hilfe von Filterkernen extrahiert. Der Filterkern wird auch Filtermatrix genannt, da der Filter in einem Matrixformat vorliegt. Bei zweidimensionalen Datensätzen kann

das Kernel eine Größe von 1x1 bis hin zur Größe der vorliegenden Matrix besitzen. Bei dreidimensionalen Datensätzen müssen die Matrizen keine dreidimensionalen Eigenschaften besitzen. (Brownlee, 2019) Die Filtermatrix wird dabei über die Pixelmatrix der vorherigen Schicht geschoben. In welcher Schrittweite das Kernel über die Bildmatrix geschoben wird, kann vom Ersteller frei gewählt werden. Eine größere Schrittweite hat einen kleineren Rechenaufwand, ein kleineres Bild nach Ausführung der Faltung, und einen eventuellen Informationsverlust zur Folge. Bei dem Schieben des Kernels wird eine pixelweise Multiplikation zwischen dem Pixel des Filters und dem dazugehörigen, vom Filter eingeschlossenen Pixel der Bildmatrix durchgeführt. Die Ergebnisse der Multiplikation zwischen Filter und Pixelwert werden anschließend addiert. Der ausgerechnete Wert wird in einer *Feature Map* gespeichert, welche die Ausgabe der errechneten Werte nach der Durchführung eines Filters darstellt. Pro Convolution Schicht, können mehrere Filter realisiert werden. Bei der Realisierung von mehreren zweidimensionalen Filtern wird die dritte Dimension der Folgeschicht, in Abhängigkeit zu der Anzahl der verwendeten Kernel, größer. Bei der Realisierung mehrerer dreidimensionaler Filter kann sich der Betrag der dritten Dimension verkleinern. (Karpathy, 2018)

Da Filter ab einer Größe von 3x3 nicht in die Pixelmatrix passen, wenn diese an Pixelposition 1,1 oder an jeder beliebigen Stelle am Rand stehen, existieren zwei Möglichkeiten diese Sonderfälle zu behandeln. Die erste Option ist die Bildmatrix mit einem *zero-padding* zu vergrößern. Durch das Zero-padding wird der Rand des Bildes solange mit 0 erweitert bis der Mittelpunkt des Filters an Position 1,1 steht und die überstehenden Zellen des Filters mit 0 gefüllt werden können. Durch die Erweiterung der Bildmatrix mit Nullen ist die berechnete Feature Map nach der Faltung genauso groß wie die Bildmatrix vor der Faltung. Die zweite Möglichkeit besteht darin, nur valide Startpunkte für den Filter zu wählen. Bei einer 3x3 Filtermatrix würde somit die erste Berechnung an Bildpunkt 2,2 anfangen, bei einem 5x5 Bild an Position 3,3 usw. Wenn die zweite Option gewählt wird, verkleinert sich die Feature Map in Abhängigkeit der Größe des Kernels. (Karpathy, 2018)

5.1.2. Pooling-Schicht

Die Pooling-Schicht dient dazu, die eben errechneten Feature Maps zu verkleinern. Durch eine Verkleinerung der Feature Map sinkt die Anzahl der zu lernenden Gewichte und der Rechenaufwand. Dadurch wird auch einem *Overfitting* des neuronalen Netzes vorgebeugt. (Karpathy, 2018)

Overfitting eines neuronalen Netzes bedeutet, dass das Netz die Beispieldaten zu gut gelernt hat und sich an diese „erinnert“. Dadurch werden gute Ergebnisse auf den Trainingsdaten erbracht aber die Klassifizierungsergebnisse von neuen Daten weisen eine nicht ausreichende Genauigkeit auf. (Brownlee, 2019)

In der Pooling-Schicht muss die Größe der Poolingmatrix angegeben werden. Jedoch werden keine Zahlenwerte, wie bei der Faltung in Kapitel 5.1.1, für die einzelnen Positionen in der Matrix eingegeben. Es wird lediglich angegeben, mit welcher Schrittweite der Poolingkernel über die errechneten Feature Maps der Convolution geschoben wird und mit welchem Verfahren das Kernel arbeitet. Es existieren drei verschiedene Pooling Verfahren.

- Average Pooling
- Max Pooling
- L2 Pooling

Average Pooling berechnet den Durchschnitt aller in der Matrixregion enthaltenen Zahlenwerte. Max Pooling sucht den größten Zahlenwert und verwirft die restlichen Werte und L2 Pooling berechnet die Summe der Kernelregion und zieht im Anschluss die Quadratwurzel. (Nielsen, 2015)

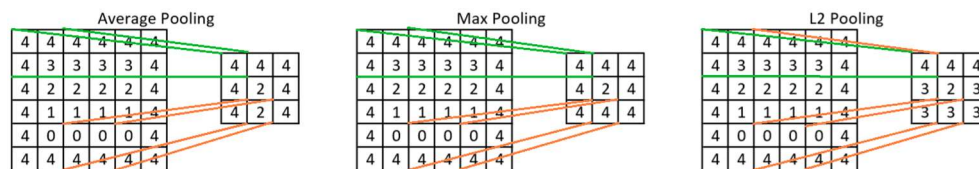


Abbildung 8: Anwendung der unterschiedlichen Pooling Strategien mit einem 2x2 Filterkernel

In obiger Abbildung soll die Ausgangsmatrix ein Bild mit schwarzem Rand und immer heller werdenden Inhalt darstellen. In den Ergebnismatrizen ist zu sehen, dass jede Poolingstrategie unterschiedliche Feature Maps erzeugt, welche unterschiedliche Erkennungsraten verursachen können.

Des Weiteren kann über das freie Festlegen der Schrittweite ein überlappendes Pooling erzeugt werden. Normalerweise wird die Schrittgröße der Poolingmatrix an deren Größe angepasst, was bei einer 3x3 Matrix die Schrittweite drei bedeuten würde. Nimmt man aber die Schrittweite zwei so überlappen sich die Pooling Matrizen mit einer Zeile und Spalte. Die Vorteile des überlappenden Pooling, im Gegensatz zum regulären Pooling, bestehen in einem kleineren Informationsverlust nach dem Pooling, bessere Erkennungsraten bei gleichbleibenden Algorithmen und Struktur sowie der verbesserten Vermeidung des Overfitting-Effekt. Die Nachteile des überlappenden Poolings sind, dass die Größe der Feature Maps nicht so schnell abnimmt, was wiederum einen erhöhten Rechenaufwand nach sich zieht. (Krizhevsky, Sutskever, & Geoffrey, 2012)

5.1.3. Vollständig Vermaschtes Netzwerk

Das vollständig vermaschte Netzwerk bildet das letzte Glied in einem CNN. In diesem Teil des neuronalen Netzes werden die errechneten Feature Maps der vorangestellten Schichten so gelernt, dass eine Klassifizierung aufgrund dieser erfolgen kann. Hierbei sind alle Neuronen einer Schicht mit allen Neuronen der

folgenden Schicht verbunden. Neuronen derselben Schicht sind jedoch unabhängig voneinander. Die Output-Schicht des vollständig vermaschten Netzwerkes stellt auch die Output-Schicht des CNN dar. Jedes Output-Neuron ist mit einer Klassifikationsgruppe gleich zu stellen, wobei aktivierte Neuronen für die Klassifizierung in diese Gruppe stehen. (Karpathy, 2018) (Brownlee, 2019)

5.2. Die ReLU-Aktivierungsfunktionen

ReLU steht für Rectified Linear Unit und steht streng genommen nicht für die Aktivierungsfunktion an sich. Es steht für den Teil des CNN's, welches die lineare Korrigierung (engl. Rectified Linear) durchführt. Da die Fachliteratur jedoch meist von der ReLU-Aktivierungsfunktion schreibt, wird hierbei diese Ungenauigkeit bewusst eingegangen, um mit den Quellen im Einklang zu bleiben und eventuellen Verwirrungen zu entgehen.

Aktivierungsfunktionen bilden den Netinput auf ein Aktivitätslevel ab. Diese Abbildung geschieht häufig in einem 2-dimensionalen Diagramm in welchem die x-Achse der Netinput und die y-Achse das daraus folgende Aktivitätslevel darstellt. (Beck & Rey, 2018)

Die am häufigsten benutzten Funktionen, neben der ReLU-Aktivierungsfunktion, sind die Sigmoid- und die Tanh-Aktivierungsfunktion. Der große Unterschied ist die beschreibende Funktion und die daraus resultierenden unterschiedlichen Aktivierungslevel des Netzes.

ReLU bringt mehrere Vorteile im Bereich des Deep Learnings, gegenüber Sigmoid und Tanh, mit sich. Der erste Vorteil ist, dass die ReLU-Funktion keinen *vanishing-gradient-effect* besitzt. Vanishing-gradient-Effekte treten durch den festen Aktivierungsintervall von (0,1) der Sigmoid oder Tanh auf. Sigmoid und Tanh projizieren große Inputs auf 1 und kleine auf fast 0, selbst wenn diese Inputs diese Werte weit übersteigen und somit wichtige Informationen darstellen würden. Außerdem sind die Funktionen in der Nähe der Extremwerte zu unsensibel. Deshalb sind die möglichen Veränderungen in diesen Bereichen, im Gegensatz zur ReLU, nur marginal möglich. Backpropagation rechnet diese marginalen Fehler durch jede vorhandene Schicht zurück, wobei der Gradient des Fehlers je Schicht kleiner wird. Dadurch haben die Neuronen in der Nähe der Output-Schicht, welche noch vernünftig große Gradienten aufweisen, einen guten Lerneffekt. Neuronen, die weiter im Inneren oder in der Nähe der Input-Schicht liegen, haben nur noch einen sehr kleinen Gradienten. Dadurch lernen diese Schichten sehr schlecht bis gar nicht. (Brownlee, 2019)

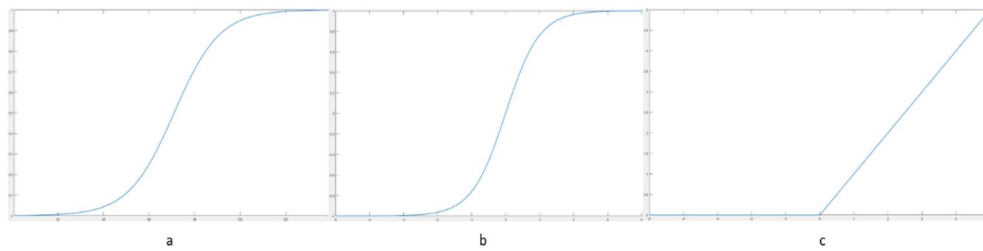


Abbildung 9: a) Sigmoid Aktivierungsfunktion
b) Tanh Aktivierungsfunktion
c) ReLU Aktivierungsfunktion

Wie in Abbildung 9 zu sehen ist, ist die ReLU Funktion eine im positiven Bereich stetige Funktion. Die erste Ableitung ist im positiven Bereich immer konstant eins und bei Ableitungen zweiten Grades immer 0, was einen großen, eindeutigen und damit gut zum Lernen geeigneten Gradienten darstellt. Der vanishing-gradient-Effekt kann somit nicht auftreten. (Goodfellow, Bengio, & Courville, 2016)

Ein weiterer Vorteil der ReLU ist, dass sie den Wert 0 annehmen kann, was bei Tanh und Sigmoid nicht der Fall ist. Xavier Glorot, Antoine Bordes und Yoshua Bengio fanden in ihrem Paper unter anderem heraus, dass die „harte“ 0 der ReLU Funktion dabei hilft das Netz genauer zu trainieren. Beim betreuten Lernen war die ReLU Funktion durchweg besser als Sigmoid, Tanh und der Softplus Funktion. Die betrachteten Testdatensätze waren MNIST, CIFAR10, NISTP und NORB.

Softplus ist angelehnt an ReLU, jedoch besitzt die Softplus keine „harte“ Null. (Glorot, Bordes, & Bengio, 2011)

5.3. Backpropagation

Folgender Text ist ein Exzerpt aus folgenden sich Überlagernde und ergänzender Quellen (Gonzales, 1996), (Nielsen, 2015). Zudem wird mit der Annahme gearbeitet, dass das verwendete neuronales Netz ein einfaches Feed-forward Netz mit einer Sigmoid-Aktivierungsfunktion ist.

Um im Genauen aufzuzeigen, welche Variablen für die Backpropagation benötigt werden, sind diese hier hintereinander aufgeführt.

- w_{jk}^l Stellt das Gewicht zwischen Neuron k der Schicht $l-1$ und Neuron j der Schicht l dar
- b_j^l Bias Neuron des j -ten Neurons der Schicht l
- a_j^l Aktivierung des j -ten Neurons der Schicht l
- z_j^l Gewichteter Input zu Neuron j in Schicht l
- δ_j^l Fehler des j -ten Neurons der Schicht l

γ Schrittweite bei der Anpassung der Gewichte

6 Sigmoid Aktivierungsfunktion

Backpropagation ist eine Möglichkeit, den an der Output-Schicht entstandenen Fehler durch das gesamte neuronale Netz zurück zu rechnen.

Der Ablauf der Backpropagation ist wie folgt:

1. Feed-forward Berechnung
2. Berechnung des Output Errors
3. Backpropagate des Fehlers durch die Schichten (Output- und Versteckte-Schichten)
4. Berechnung des Gradienten
5. Anpassen der Gewichte

Im ersten Schritt werden dem Netz die verschiedenen Testdaten präsentiert. Das Netz gibt den Inputvektor Schicht für Schicht durch das neuronale Netz und errechnet so einen Outputvektor. Dabei unterscheiden sich der errechnete Outputvektor von dem gewünschten Outputvektor.

Die Fehlerberechnung der Output-Schicht, welche Schritt zwei darstellt, benötigt zur Berechnung die Kostenfunktion und den gewichteten Input.

$$C = \frac{1}{2n} * \sum_x ||y(x) - a^L(x)||^2$$

Formel 5: Kostenfunktion

n ist die Anzahl der verwendeten Testdaten, x ist ein einzelner Testdatensatz, $y(x)$ ist der gewünschte Outputvektor für das Trainingssample x und a^L ist die Aktivierung der Schicht L . L stellt die Anzahl der Schichten im Netz dar, wodurch a^L der momentane Output des Netzes ist.

Die Kostenfunktion ist jedoch mit zwei Annahmen versehen. Annahme eins ist, dass die Kostenfunktion als Durchschnitt über alle Testdaten dargestellt werden kann. Diese Annahme ist umsetzbar, da die Formel $C = \frac{1}{n} * \sum_x C_x$ mit der Kostenfunktion für Testsample x $C_x = \frac{1}{2} * ||y - a^L||^2$ die getroffene Annahme erfüllt. Die Fehlerfunktion für Backpropagation lautet wie folgt.

$$\delta_j^L = \frac{dC}{da_j^L} * \sigma'(z_j^L)$$

Formel 6: Fehlerfunktion für ein einzelnes Neuron

Optional kann die Fehlerfunktion für Matrizen zur Berechnung genommen werden.

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

Formel 7: Fehlerfunktion in Matrixschreibweise

Die folgende Gleichung, welche den Vorgang der Backpropagation darstellt, baut auf eben vorgestellter Gleichung 7 auf.

$$\delta^l = ((w^{l+1})^T * \delta^{l+1} \odot \sigma'(z^l)$$

Formel 8: Fehler der nächsten Schicht

Formel 8 ist essenziell für die Backpropagation. Die Fehlerfunktion für den Fehler der nächsten Schicht ist trivial. Wenn das NN den Backpropagation Vorgang starten, liegen dem Netz bereits folgende Informationen vor. w^{l+1} sind die Gewichte der nächsten Schicht. δ^{l+1} ist bei der ersten Anwendung der Formel die Fehlerfunktion des NN, was das Ergebnis aus Gleichung 7 ist. Beim zweiten Anwenden von Formel 8 ist δ^{l+1} das Ergebnis der ersten Anwendung von Formel 8. Durch diesen Zusammenhang rechnet sich die Gleichung durch das gesamte Netz zurück. Wenn sich die Formel durch das gesamte Netz gearbeitet hat, liegt der Fehler für jede Schicht in Vektorschreibweise vor. Die Aktivierungsintensität der Neuronen aus Schicht l wird mit $\sigma'(z^l)$ berechnet.

Der in Formel 7 und 8 enthaltene Parameter $\sigma'(z^l)$ zeigt auch auf, wie schnell ein Netz lernt. Dies wird in Abbildung 10 aufgezeigt. In der Abbildung ist die erste Ableitung der Sigmoid Aktivierungsfunktion dargestellt, welche aufzeigt, wie gut ein Neuron bei bestimmten Aktivierungen lernt. Dabei ist zu sehen, dass der Anstieg in der Nähe von null und eins, gegen 0 geht. Der geringe Anstieg in diesen Bereichen stellt eine „Sättigung“ des Neurons dar. Wenn ein Neuron „gesättigt“ ist, lernt es nur noch langsam oder, wie bei der ReLU-Funktion durch die „harte“ 0, gar nicht mehr.

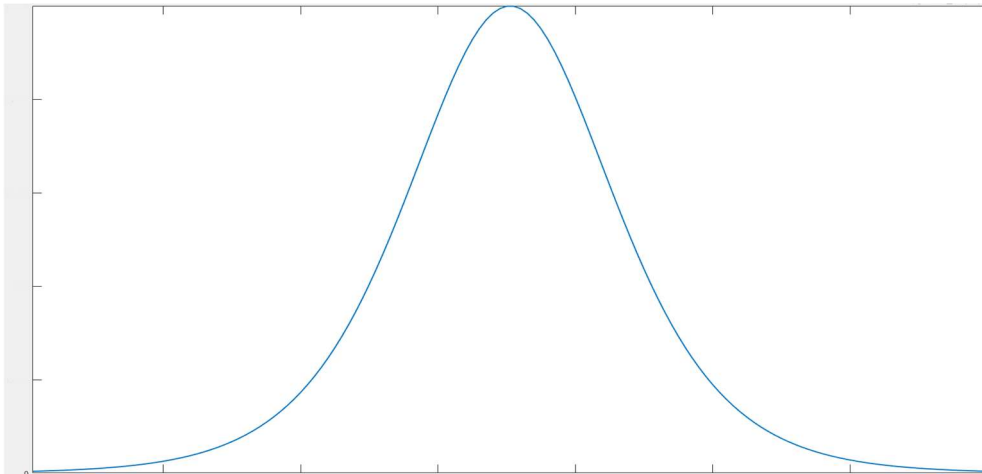


Abbildung 10: Erste Ableitung der Sigmoid Aktivierungsfunktion

Die letzten beiden Punkte im Ablauf der Backpropagation geschehen in einer Rechnung.

$$\Delta w_{ij} = -\gamma * a_i^{l-1} \delta_j^l$$

Formel 9: Formel zum Ändern der Gewichte

γ stellt die vom Benutzer gewählte Schrittweite beim Lernen dar und $a_i^{l-1} \delta_j^l$ ist der Gradient der Fehlerfunktion. Aus diesem Grund wird die Schrittweite auch negativ mit dem Gradienten multipliziert, um dem Fehler entgegen zu wirken.

5.4. Segmentation Algorithmen

Die Klassifizierung von Bildern unterscheidet sich von der Objekterkennung in Bildern. Bei der Klassifizierung wird jedem Bild nur einer Gruppe zugeordnet. Bei der Objekterkennung kann jedoch jedes Bild mehrere Objekte aus mehreren Gruppen enthalten. Um die einzelnen Objekte zu erkennen und deren Position zu bestimmen muss das Bild segmentiert werden, um anschließend eine Bounding Box um diesen Bereich ziehen zu können. (Zhang, C. Lipton, Li, & J. Smola, 2020)

Dieses Kapitel wird die drei Segmentationsalgorithmen YOLO, SSD und Faster R-CNN betrachtet. Es werden nur die drei Algorithmen betrachtet, da sie die am meisten verbreiteten und unter anderem mit zu den schnellsten Algorithmen zur Segmentierung zählen. (Redmon & Farhadi, 2018) Aus Gründen der fehlenden Rechenleistung des Erstellers dieser Arbeit wird auf andere und komplexere Verfahren verzichtet.

5.4.1. You only look once (YOLO)

YOLO ist der erste der drei Algorithmen zum Segmentieren eines Bildes. Aufgrund dieser Segmentierung kann ein Objekt in einem Bild erkannt, klassifiziert und die Position mit Hilfe einer *Bounding Box* bestimmt werden. Zuerst wird das Bild in ein $S \times S$ -Grid unterteilt. Darauf werden in jedem Bereich des Grids B Bounding Boxes aufgespannt. Dabei kann frei bestimmt werden, wie viele Bounding Boxes pro Grid erzeugt werden sollen.

Jede vom Netz aufgespannte Bounding Box erhält einen *Confidence Score* für ein bestimmtes Objekt. Der zugewiesene Score bestimmt nicht nur wie wahrscheinlich es ist, dass ein Objekt in der aufgespannten Bounding Box enthalten ist, sondern auch die Einschätzung des Netzes über die Genauigkeit der Box.

Die Bounding Boxes enthalten dabei fünf Parameter. Der x und y Parameter bestimmen den Mittelpunkt der Box. Diese zwei Parameter stehen im Kontext des Grids. w und h sind die Höhe und Breite der Box im Kontext des gesamten Bildes. Der fünfte Wert ist der oben beschriebene Confidence Score.

Parallel dazu wird jedem Bereich des Grids einer Klasse C zugeordnet. Jeder Bereich kann jedoch nur eine Klasse zugeordnet werden. Nachdem das gesamte

Gitter des Bildes unterteilt ist, besteht das Bild idealerweise aus unterschiedlich klassifizierten Bereichen.

Wenn die Berechnungen der Bounding Boxes und der Gridklassifizierung erfolgt sind, werden die Werte miteinander multipliziert. Aus diesem Wert werden dann die Bounding Boxes im gesamten Bild berechnet. (Redmon, Divvala, Girshick, & Farhadi, 2016)

YOLO ist der schnellste von den vorzustellenden Algorithmen Segmentierungsalgorithmus. YOLOv3 ermöglicht es, Bounding Boxes in Echtzeit zu erzeugen. Die maximal mögliche Bilderrate, bei einem 320x320 Pixel großen Bild und einer TitanX Grafikkarte, beträgt 45 FPS. Die Genauigkeit der Positionsbestimmung ist genauso gut wie der, im folgenden Kapitel vorgestellten, Algorithmus SSD und nur etwas schlechter als das Faster R-CNN aus Kapitel 5.4.3.. (Redmon & Farhadi, 2018)

Das im April 2020 erschienene YOLOv4 ist die derzeit aktuellste YOLO-Version. Version 3 und Version 4 sind ungefähr gleich schnell. In der Genauigkeit unterscheiden sich die beiden Versionen jedoch enorm. Die *Average Precision* von YOLOv4 ist in allen Fällen um 20%-30% höher als die von YOLOv3. (Bochkovskiy, Wang, & Mark Liao, 2020)

5.4.2. Single Shoot Detector (SSD)

SSD arbeitet wie YOLO indem es ein Gitter über das Bild legt, welches das Bild in gleich große Teile unterteilt. Für jeden Bereich des Bildes werden verschieden große und frei wählbare *Anchor Boxes* deklariert. Jede Box besitzt dabei $K+1$ Kategorien. K steht hierbei für die Anzahl der Klassifikationsgruppen und $+1$ weil eine Anchor Box ohne Objekt mit 0 kategorisiert wird. (Zhang, C. Lipton, Li, & J. Smola, 2020) Die Anchor Boxes können jede beliebige rechteckige Form besitzen. Ein Zoom Level bestimmt dabei wie groß die Anchor Box, im Verhältnis zu einem Bereich des Gitters, ist. Jede Form hat dabei Einfluss auf die Erkennungsrate des Netzes. Eine Box mit den Maßen 4x2 (4 breit und 2 hoch) ist schlechter zur Erkennung von aufrecht laufenden Menschen geeignet als eine 2x4 Anchor Box.

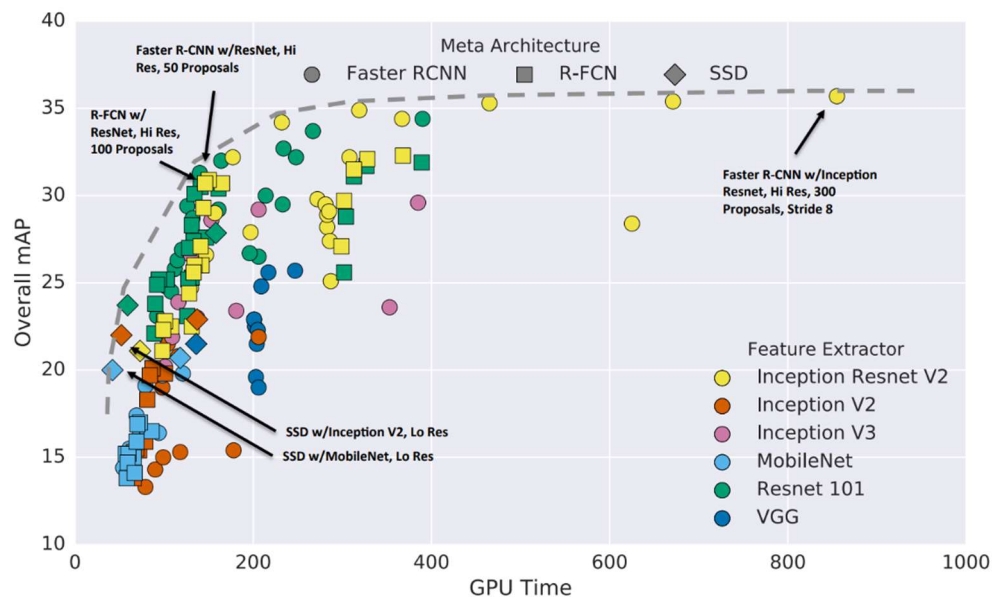


Abbildung 11: Ergebnis eines Versuches in welchem zwischen Genauigkeit und Rechenzeit abgewägt wird. Die Form gibt die Art des verwendeten Netzes an und die Farbe die Art der Merkmalsextraktion. Quelle: (Huang, et al., 2017), S. 8

Die Geschwindigkeit von SSD ist ein wichtiger Faktor, ob für das CNN SSD verwendet wird oder nicht. Wie obige Abbildung erkennbar macht, ist SSD in allen Fällen schneller als das im folgenden Kapitel vorgestellte Faster R-CNN. Jedoch ist die durchschnittliche Genauigkeit der erzeugten Bounding Boxes geringer. Im Vergleich zu YOLOv4 weist es jedoch einige Schwächen auf. Die Rechenzeit von SSD ist im Gegensatz zu YOLO um circa das 2-3-fache höher. Jedoch ist die Genauigkeit zu YOLO v3 nur um 1-5 Punkte höher und zu der im April 2020 erschienenen YOLO v4 sogar geringer.

5.4.3. Faster Region-Based Convolutional Neuronal Network (Faster R-CNN)

Faster R-CNN's verwenden ein vorgestelltes CNN, welches eine Feature Map von dem zu klassifizierenden Bild erstellt. Diese Feature Map wird für zwei Dinge verwendet. Die erste Anwendung besteht in der Vorbestimmung von möglichen Objekten in einem *Region Proposal Network* (RPN) und die zweite aus dem Region of Interest-Pooling, auch RoI-Pooling genannt. Dieses skaliert die Größe der Feature Map auf die geforderte Größe der folgenden Klassifizierungsschichten. (Grel, 2017)

Das RPN verwendet die Feature Maps des CNN's und gibt rechteckige Bereiche des Bildes mit einer dazugehörigen Wahrscheinlichkeit, zu welcher Gruppe dieser Bereich gehört, zurück. Um dies zu erreichen, wird ein $n \times n$ große - Schicht erstellt, welche sich über die Feature Map schiebt. Das Ergebnis wird in einer Box-Regression-Schicht und einer Box-Klassifizierungs-Schicht bearbeitet. Die Box-Regression-Schicht ist für die Anpassung der Größe der Bounding-Box zuständig und die Box-Klassifizierungs-Schicht wird für die Klassifizierungswahrscheinlichkeit benötigt. Die zwei Schichten werden üblicherweise mit einer 1×1 Convolutional Kernel umgesetzt. (Grel, 2017) (Ren, He, Girshick, & Sun, 2016)

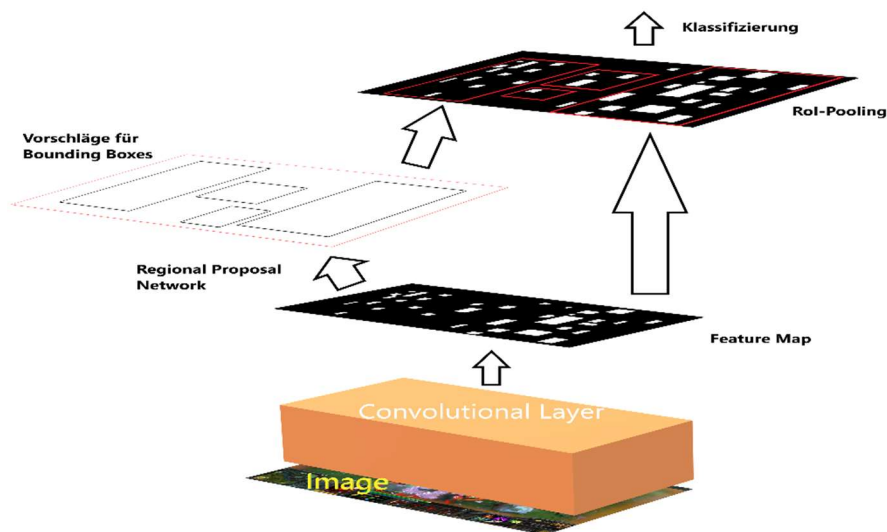


Abbildung 12: Aufbau eines Faster R-CNN Quelle (Ren, He, Girshick, & Sun, 2016) nachgestellt.

Im Anschluss übernimmt die Klassifizierungs-Schicht, welche nach dem RoI-Pooling implementiert wird, die Bestimmung von genauen Bounding-Boxes und berechnet die Wahrscheinlichkeit zu welcher Gruppe diese Box gehört.

Abbildung 12 zeigt einen solchen Aufbau eines Faster R-CNN noch einmal auf. Das Faster R-CNN ist der langsamste aber genaueste der drei vorgestellten Algorithmen, was in Abbildung 11 erkennbar wird. (Huang, et al., 2017) Die geringere Geschwindigkeit und die höhere Genauigkeit der Bounding-Boxes entsteht durch die Arbeitsweise des Netzes. Durch eben beschriebene Arbeitsweise wird klar, dass Faster R-CNN's das Bild zweimal verarbeiten, um die endgültigen Bounding-Boxes zu bestimmen. YOLO und SSD gehen hingegen nur einmal über das Bild.

Dadurch wird Zeit und Rechenleistung gespart. Des Weiteren ist der Aufbau eines Faster R-CNN komplexer als bei den beiden anderen

Segmentierungsalgorithmen, was wiederum die Rechenzeit und den Speicherbedarf anhebt. (Huang, et al., 2017)

6. Realisierung der Patternerkennung mittels einem Convolutional Neuronal Network

Nach dem in den vorherigen Kapiteln Grundlagen der einzelnen Bestandteile zur Patternerkennung mit einem CNN erklärt wurden, geht es in diesem Abschnitt um die Realisierung der einzelnen Zwischenschritte.

Zu aller erst müssen auf das gewünschte Musikstück ein Low-Pass-Filter, ein Band-Pass-Filter und ein High-Pass-Filter gelegt werden. Durch die Verwendung von drei Filtern können mehr Informationen aus einem einzelnen Spektrum gewonnen werden. Aus diesem Grund werden später pro Musikstück drei Klassifizierungen durch das Netz nötig sein. Danach werden die Spektren des Low- und High-Pass- an ein neuronales Netz gegeben, um die Pattern des Basses, der Bass-Drum und der Clap zu erkennen. Die Bass-Drum und die Clap sind, wie Abbildung 13 a und c beweist, mit dem bloßen Auge erkennbar.

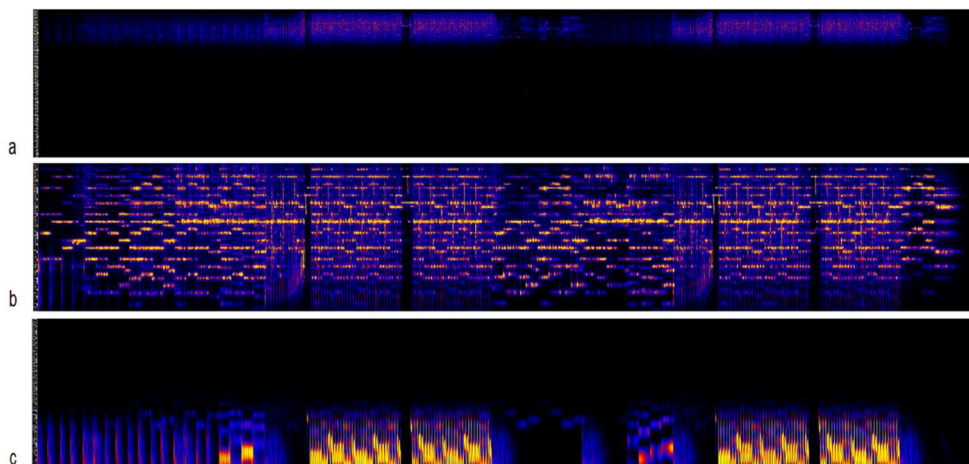


Abbildung 13: a) Spektrum Highpass-Filter
b) Spektrum Bandpass-Filter (Zoomed in)
c) Spektrum Lowpass-Filter

Das Spektrum des Middle-Pass-Filters wird der aufwendigste Teil in dieser Klassifikation. In diesem Frequenzbereich befinden sich hauptsächlich die Frequenzen der chromatischen Tonleiter. (Gedan, 2006) Somit sind in dem Bereich alle restlichen Instrumente enthalten. Dadurch überlagern sich in diesem Spektrum auch die Frequenzen der Instrumente, was das Spektrum in Abbildung 13 b verdeutlicht. Aus diesem Grund wird es nicht mehr möglich sein, die erkannten Pattern einem Instrument zuzuordnen. Somit soll das CNN lediglich wiederholende Muster im Spektrum erkennen. Zuletzt werden die erkannten Pattern mithilfe von Bounding Boxes angezeigt.

6.1. Realisierung des Low-/Band-/High-Pass-Filters

Für die Umsetzung der drei Filter wird das aus der Python Bibliothek stammende Packet *aubio* verwendet. Dieses ist unter anderem zum Einlesen der *.wav* zuständig. Da es in diesem Paket keine Funktionen für einen Low-/Band-/High-Pass-Filter gibt, werden diese mit einem Biquad-Filter realisiert. Ein Biquad-Filter ist ein *infinite impulse response* Filter der zweiten Ordnung, welcher als Grundbaustein für komplexere Filter dient. (Redmon N. , Biquads, 2003)

Als nächstes müssen die Koeffizienten für die Biquads bestimmt werden. Dafür werden mehrere Online Rechentools für die Berechnung der Biquad-Koeffizienten verwendet und die Ergebnisse im Anschluss verglichen, um eine fehlerhafte Rechnung auszuschließen. Die verwendeten Rechentools sind von Arachnoid (Lutus, 2019), EasyCalculation.com (EasyCalculation.com) und Ear Level Engineering (Redmon N. , 2013).

Die Kurvenverläufe der Dämpfung des Filters müssen dabei den Anforderungen hingehend bestimmt werden. Der Low-Pass-Filter muss Frequenzen bis 150 Hz möglichst klar darstellen und Frequenzen über 200 Hz komplett herausfiltern. Der Band-Pass-Filter soll Frequenzen ab 200 Hz darstellen bis zu einer Frequenz von 6000 Hz, da in diesem Frequenzbereich die chromatische Tonleiter aller Instrumente liegt. (Gedan, 2006) Der überbleibende Frequenzbereich von 6000 Hz bis 21000 Hz muss vom High-Pass-Filter abgedeckt werden.

Für den Low-Pass-Filter sind folgende Angaben getroffen worden:

- Samplerate: 41000
- Mittenfrequenz: 150 Hz
- Bandbreite 0,75

Berechnet wurden folgende Koeffizienten:

- $a_0=1$
- $a_1=-1,96929513$
- $a_2=0.96981555$
- $b_0=1,30104102 \cdot 10^{-4}$
- $b_1=2,60208205 \cdot 10^{-4}$
- $b_2=1,30104102 \cdot 10^{-4}$

Der Band-Pass-Filter ist aus einem High-Pass- und einem Low-Pass-Filter zusammengesetzt, wodurch folgende Angaben für den Filter getroffen wurden:

Low-Pass-Filter:

- Samplerate: 41000
- Mittenfrequenz: 6000 Hz
- Bandbreite 0,7

High-Pass-Filter:

- Samplerate: 41000
- Mittenfrequenz: 200 Hz
- Bandbreite 0,7

Berechnet wurden folgende Koeffizienten:

- $a_{0_l}=1$
- $a_{1_l}=-0,77321399$
- $a_{2_l}=0,27545624$
- $b_{0_l}=0,12556056$
- $b_{1_l}=0,25112113$
- $b_{2_l}=0,12556056$
- $a_{0_h}=1$
- $a_{1_h}=-1,95624013$
- $a_{2_h}=0,95715934$
- $b_{0_h}=0,97834987$
- $b_{1_h}=-1,95669974$
- $b_{2_h}=0,97834987$

Für den High-Pass-Filter sind folgende Angaben getroffen worden:

- Samplerate: 41000
- Mittenfrequenz: 6000 Hz
- Bandbreite 0,7

Berechnet wurden folgende Koeffizienten:

- $a_0=1$
- $a_1=-0,77321399$
- $a_2=0,27545624$
- $b_0=0,51216756$
- $b_1=-1,02433511$
- $b_2=0,51216756$

Nach Bestimmung der Koeffizienten sind folgende Filterfunktionen realisiert worden.

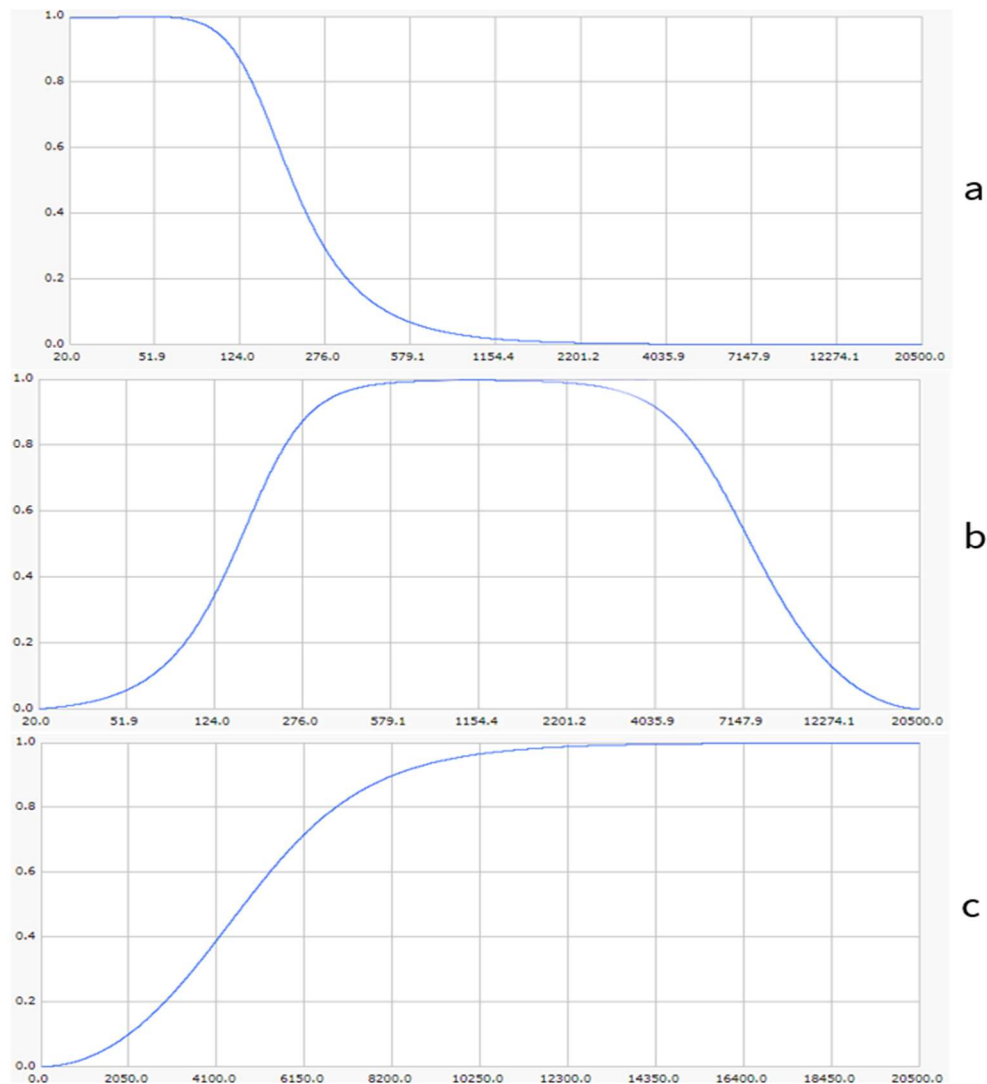


Abbildung 14: Kurvenverläufe der einzelnen Filter, wobei die x-Achse die Frequenzen und die y-Achse die Dämpfung darstellen.

a) Low-Pass-Filter

b) Band-Pass-Filter

c) High-Pass-Filter

Da, wie oben zu sehen, folgende Funktionen keine scharfen Kanten an den Grenzen aufweisen, werden mehrere Biquads mit gleichem Koeffizienten hintereinander implementiert. Das führt dazu, dass die Grenzen der Filter schärfer werden. (Redmon N. , Biquads, 2003)

Nachdem die Filter auf das Musikstück angewendet wurden, bewiesen Hörproben, dass die Filter funktioniert haben. Ob die Grenzen an den gewünschten Frequenzen liegen, lässt sich durch das Erzeugen der zugehörigen Spektren beweisen.

6.2. Erzeugen der Spektren

Nachdem alle Filter angewendet wurden, liegen nun drei MP3-Dateien pro Musikstück vor. Da jedoch dem CNN die Spektren der einzelnen Dateien vorgelegt werden sollen, müssen diese erst erstellt werden. Dafür wird das Programm *Sonic Visualizer* verwendet.

Sonic Visualizer ist ein Open Source Programm der Queen Mary Universität London, mit welchem Musik mittels einer Vielzahl von Tools analysiert werden kann. (Cannam, Landone, & Sandler, 2019) Die erzeugten Spektren der MP3-Dateien sind vergleichbar zu Abbildung 13 a,b und c.

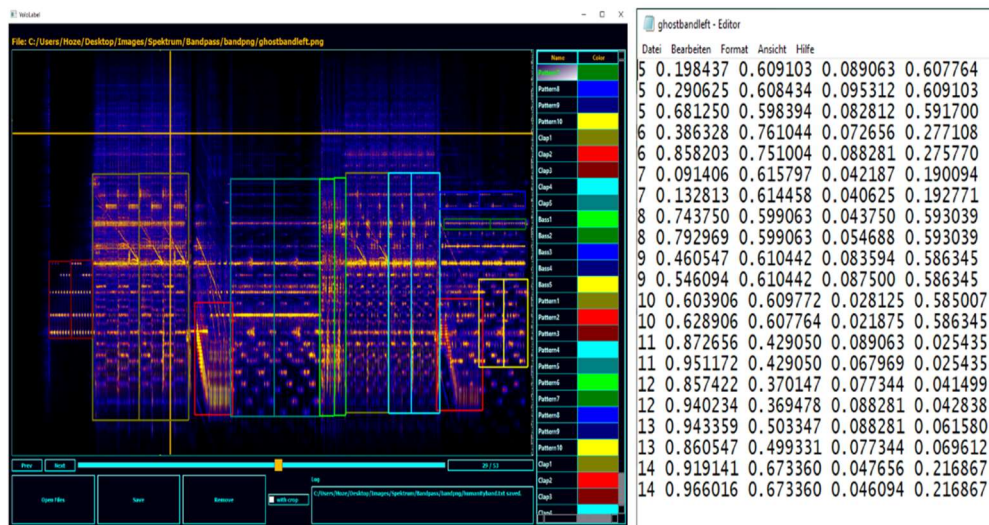
Bei der Kontrolle der exportierten Bilder der Spektren sind die Grenzen der Filter klar erkennbar. Das Spektrum des Low-Pass-Filters befindet sich im geforderten Bereich bis 200 Hz. Bis zu 300 Hz sind einige Fragmente des Musikstückes vorhanden, aber diese sind so gering, dass diese bei der Auswertung nicht ins Gewicht fallen. Auch der Band-Pass-Filter und der High-Pass-Filter weisen die geforderten Grenzen von 200Hz-6000Hz bzw. 6kHz-21kHz auf. Aus diesem Grund kann davon ausgegangen werden, dass alle Filter aus Kapitel 6.1. wie gewünscht funktioniert haben. Mit den exportierten Bildern ist die erste Voraussetzung für das Erkennen der Pattern in diesem Musikstück mit einem CNN gegeben. Die nächste Aufgabe ist das Erstellen des CNN.

6.3. Labeln der Erzeugten Spektren

Nachdem alle Spektren erzeugt wurden, müssen diese für das Neuronale Netz gelabelt werden. Bilder müssen gelabelt werden, um ein NN beim Lernen zu unterstützen. Dabei werden die zu erkennenden Objekte in einem Bild mit rechteckigen Bounding Boxes markiert. Die für diese Arbeit erstellten Bilder wurden mittels der Software *Yolo Label* gelabelt. Die Software ist unter folgenden Githublink zu finden: https://github.com/developer0hye/Yolo_Label

Diese erzeugt für jedes Bild eine .txt-Datei, in welcher Zeilenweise folgende Informationen stehen: <Objektklasse> <Mittelpunkt x> <Mittelpunkt y> <Breite> <Höhe>

Folgende Abbildung zeigt den Vorgang des Labelns und die daraus resultierende Datei.



a

b

Abbildung 15: a) Erstellen der Labels mit YoloLabel (Einzelbild siehe Anhang)
b) Daraus resultierende Textdatei

Die letzten vier Werte sind keine festen Pixelwerte, sondern relative Angaben. Somit ist der Wert für Mittelpunkt x einer Bounding Box bei einem 120px großen Bild nicht 40, sondern 0.333333. (Bochkovskiy, Github, 2020)

6.4. Erstellen des Convolutional Neural Networks

Bevor das Netz erstellt werden kann, muss abgewägt werden, welcher Segmentationsalgorithmus sich am besten für die Erkennung der Strukturen eignet. Dabei musste ein Kompromiss zwischen Rechenleistung und Genauigkeit getroffen werden. Dabei ist die Entscheidung auf YOLOv4 von Alexey Bochkovskiy, Chien-Yao Wang und Hong-Yuan Mark Liao gefallen. Die Wahl für YOLOv4 wurde aufgrund der hohen Genauigkeit und den kurzen Berechnungszeiten gefällt. Die Berechnungszeiten sind so gut, dass die Klassifizierung theoretisch in Echtzeit in bis zu 120 FPS möglich ist. (Bochkovskiy, Wang, & Mark Liao, 2020)

Für die Erstellung des CNN wird das Framework *darknet* von Joseph Redmon und YOLOv4 verwendet. Darknet ist ein open source Framework für Neuronale Netze, welches in C und CUDA geschrieben ist.

Da eine genaue Schritt für Schritt Anleitung zum Erstellen von Darknet und zum Trainieren und Testen des Netz nicht Bestandteile dieser Arbeit sind, wird hier auf die GitHub-Seite des Erstellers von YOLOv4 verwiesen: <https://github.com/AlexeyAB/darknet>. Somit werden nur die Schritte zum

Erstellen von *darknet*, Anpassen von Dateien für das Lernen und die Befehle zum Trainieren und Testen näher erläutert.

Unter Linux ist die Erstellung des Netzwerks trivial. Es muss das Git-Repository geklont und mittels *make* in der Konsole erstellt werden. Im Anschluss müssen zwei Textdateien, in welchem die Test- und Evaluierungsdaten aufgelistet sind, eine *.cfg*, eine *.data*, und eine *.names* Datei angepasst werden.

Die erzeugten Spektren wurden für dieses Netz einmal normal und einmal gespiegelt gelabelt, da so ein größerer Lernpool für das Netz zur Verfügung steht. Für dieses Netz wurden 26 verschiedene Musikstücke gelabelt. Da jedes Musikstück in drei Spektren unterteilt wurde, ergeben sich somit 78 Spektren. Durch die Spiegelung jedes Spektrums entsteht ein Lernpool aus 156 Spektren, wobei 125 Bilder zum Lernen und 31 zum Validieren genutzt wurden.

Mit folgendem Befehl wird das Netz trainiert und erstellt: `./darknet detector train <Pfad zur .cfg Datei> <Pfad zur .data Datei> <Pfad zu Datei mit den Startgewichten>`

Das Trainieren dauert einige Stunden bis zu einigen Tagen, je nachdem, ob eine GPU oder CPU zur Berechnung verwendet wird und je nach Größe der eingegebenen Bilder, maximalen Batches und der Anzahl zu erkennende Klassen. (Bochkovskiy, Github, 2020)

Um das Netz nach dem Lernen zu testen, muss folgender Befehl (in Linux) eingegeben werden: `./darknet detect <Pfad zur .cfg Datei> <Pfad zur Datei mit trainierten Gewichten> <Pfad zum Bild welches Erkannt werden soll>`

Welche Ergebnisse mit diesem Netz erreicht wurden, ist in Kapitel 9 vorzufinden.

7. Patternerkennung mithilfe von Rhythmus und Melodie

Nachdem in Kapitel 6 die Spektren der einzelnen Musikstücke für die Erkennung von Pattern benutzt wurden, werden in diesem Kapitel die Noten der Musikstücke für die Erkennung von Pattern verwendet. Die in diesem Abschnitt beschriebenen Algorithmen arbeiten entweder mit den gespielten Noten des Musikstückes oder mit einem fünfdimensionale-Set (Kapitel 7.3.1.) der Noten. Bei der String basierte Patternerkennung und die Patternerkennung mithilfe von Matrizen werden die Musikstücke mittels MIDI-Dateien eingelesen. In den MIDI-Dateien sind unter anderen Informationen zu den gespielten Noten zu finden. Die Informationen beinhalten die Tonhöhe, Lautstärke und Dauer der Note. Die Zeit, wann eine gewisse Note gespielt wird ist ebenfalls implizit gegeben. (MIDI Association, 2019) Diese Information wird aber nicht verwendet, da beide Algorithmen durch das implementierte *Pattern Matching* ausschließlich auf die Reihenfolge der gespielten Noten achtet und nicht auf eine zeitliche Abhängigkeit.

Alle in dieser Arbeit vorgestellten Algorithmen, welche mit der Geometrischen Patternerkennung arbeiten, verwenden ein fünftdimensionales Punkt Set. Das Punkt Set wird im nächsten Schritt so verarbeitet, dass es in ein zwei- oder dreidimensionales Koordinatensystem eingetragen werden kann. (Meredith, 2006)

7.1. String basierte Patternsuche

Die String basierte Patternsuche wird in Python umgesetzt. Zum Auslesen der MIDI-Dateien wird das Package *mido* verwendet. Nachdem die MIDI-Dateien ausgelesen sind, werden die einzelnen Tracks, die in einem MIDI-File existieren, in ein Array gespeichert und an den eigentlichen Algorithmus zum Finden der Pattern weitergeleitet. Wenn die MIDI-Werte in dem Array gespeichert sind, ist der zeitliche Aspekt verloren gegangen, da Noten, die gleichzeitig gespielt werden, nacheinander in dem Array stehen. Somit kann ein Pattern womöglich nicht erkannt werden, da die *note_on*-Message in der MIDI-Datei für zwei gleichzeitig gespielte Noten in unterschiedlichen Varianten in der MIDI-Datei vorkommen. Der Algorithmus berechnet zuerst die maximale Patternlänge. Diese beträgt $\lfloor \text{Länge des Trackarrays} / 2 \rfloor$. Danach wird mit jeder Länge einmal das Musikstück iterativ durchgerechnet. Die Iteration beinhaltet die Änderung der Startposition des Patterns im Track um eine Stelle. Angefangen wird mit dem größtmöglichen Pattern bis zu einer Patternlänge von 3 Noten. Wenn der Algorithmus gestartet wird, ist das im Moment zu untersuchende Pattern im Intervall der aktuellen Position und *aktuelle Position* + *Patternlänge* zu finden. Im Anschluss wird die erste Stelle des Patterns mit der Position *erste Stelle des Patterns* + *Patternlänge* verglichen. Wenn die beiden Positionen übereinstimmen, wird die zweite Position des Patterns mit der Position *erste Stelle des Patterns* + *Patternlänge* + 1 verglichen und das solange bis die beiden Stellen nicht mehr übereinstimmen. Wenn jedoch die beiden Stellen nicht übereinstimmen, wird in den Fall des Keychange oder der vertikalen Spiegelung gegangen. In diesen beiden Fällen werden sowohl die aktuelle Position als auch die nächsten beiden Positionen mit den dazugehörigen Positionen verglichen.

Für ein tieferes Verständnis des Algorithmus, kann das im Anhang befindende Bild 2 angeschaut werden.

Bei einem Keychange werden nur die Abstände zwischen der ersten/zweiten/dritten Position des Patterns und der aktuellen/nächsten/übernächsten Position im Track verglichen. Sollten alle drei Positionen denselben Abstand aufweisen, wird ein Zähler auf 3 gesetzt, um alle diese Noten in das vorhandene Pattern aufzunehmen. Die vertikale Spiegelung verläuft ähnlich zum Keychange, nur dass bei der vertikalen Spiegelung eine gemeinsame Spiegelachse vorliegen muss. Diese wird gesucht, indem die zwei MIDI-Werte der aktuellen Position und der ersten Stelle des Patterns miteinander addiert werden und im Anschluss durch 2 geteilt werden. Sollte der eben

errechnete Wert mit den Spiegelachsen der nächsten zwei Positionen übereinstimmen, so liegt eine vertikale Spiegelung vor.

Nachdem der Algorithmus mit jedem Track eines MIDI-Files fertig ist, werden die gefundenen Pattern in eine .txt Datei geschrieben. In dieser wird angegeben, welches Pattern gefunden wurde, welche Länge das gefundene Pattern aufweist und an welcher Stelle im Track es gefunden wurde. Mit diesen drei Eigenschaften kann der Anfang des Patterns problemlos gefunden werden.

Welche Ergebnisse die String basierte Patternsuche gegenüber den anderen Arten der Patternerkennung erzielt, wird in Kapitel 9 aufgezeigt.

7.2. Patternerkennung mithilfe von Matrizen

Die Patternerkennung mit Matrizen, wurde wie auch schon die Patternerkennung mit Strings in Python umgesetzt. Die MIDI-Dateien werden abermals mittels *mido* ausgelesen. Dabei ist abermals der zeitliche Aspekt verloren gegangen in derselben Weise wie in Kapitel 7.1.

Bei der Umsetzung mit Matrizen wird in die erste Zeile und in die erste Spalte des momentan zu bearbeitendem Track geschrieben. Im nächsten Schritt wird die Matrix diagonal an den Zellen $x = y$ unterteilt. Die untere Hälfte der Matrix ist dabei nicht verwendbar, da dies einem Abgleich in einen früheren Teil des Liedes gleichkommen würde. Die untere Hälfte würde ebenfalls nur eine Spiegelung der oberen Hälfte darstellen. Ein Beispiel für eine solche Matrix ist in folgender Abbildung gegeben. (Hsu & Chen, 1998)

| | 49 | 52 | 60 | 49 | 52 | 60 | 52 | 55 | 62 |
|----|----|----|----|----|----|----|----|----|----|
| 49 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 52 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 2 | 0 |
| 60 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 3 |
| 49 | 1 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 |
| 52 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 5 | 0 |
| 60 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 6 |
| 52 | 1 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 |
| 55 | 0 | 2 | 0 | 0 | 5 | 0 | 0 | 0 | 0 |
| 62 | 0 | 0 | 3 | 0 | 0 | 6 | 0 | 0 | 0 |

a

| | 49 | 52 | 60 | 49 | 52 | 60 | 52 | 55 | 62 |
|----|----|----|----|----|----|----|----|----|----|
| 49 | - | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 52 | | - | 0 | 0 | 2 | 0 | 0 | 2 | 0 |
| 60 | | | - | 0 | 0 | 3 | 0 | 0 | 3 |
| 49 | | | | - | 0 | 0 | 4 | 0 | 0 |
| 52 | | | | | - | 0 | 0 | 5 | 0 |
| 60 | | | | | | - | 0 | 0 | 6 |
| 52 | | | | | | | - | 0 | 0 |
| 55 | | | | | | | | - | 0 |
| 62 | | | | | | | | | - |

b

Abbildung 16: Matrix a stellt die komplette Berechnung der Matrix dar. In Matrix b ist die effizientere Variante zu sehen, da die Berechnung der unteren Hälfte ausgespart werden kann.

Wenn die Matrix initialisiert ist, kann der Algorithmus mit der Berechnung anfangen. Dabei wird durch jede Zelle der Matrix gegangen und analog zu der Patternerkennung mithilfe von Strings bearbeitet. Bei gleichen MIDI-Werten der

Zellenindizes einer Zelle (x,y) erhält diese den Wert der Zelle(x − 1, y − 1) + 1. Somit wird für eine Wiederholung Zelle(x, 0) und Zelle(0, y) auf Gleichheit überprüft.

Bei einem Keychange muss folgende Aussage wahr sein:
 $Zelle(0, y) - Zelle(x, 0) == Zelle(0, y + 1) - Zelle(x + 1, 0) ==$
 $Zelle(0, y + 2) - Zelle(x + 2, 0)$. Eine vertikale Spiegelung liegt vor, wenn die Aussage

$$\frac{Zelle(0,y)+Zelle(x,0)}{2} == \frac{Zelle(0,y+1)+Zelle(x+1,0)}{2} == \frac{Zelle(0,y+2)+Zelle(x+2,0)}{2}$$
wahr ist. Wenn die Berechnung der Matrix vollendet ist, können die Pattern aus der Matrix gelesen werden. Die Länge des Patterns ist durch den Zelleninhalt am weitesten rechts unten einer zusammenhängenden diagonalen Kette gegeben. Die Zeilenindizes am linken oberen Ende einer Kette stellen die Anfangspunkte für das Pattern dar. Abbildung 16 macht diesen Sachverhalt noch einmal deutlich.

Diese Informationen werden in eine .txt Datei geschrieben, wobei deren Aufbau vergleichbar zum Aufbau der Ergebnisdatei aus Kapitel 7.1 ist.

7.3. Vektorgeometrische Patternsuche

7.3.1. Fünf-Dimensionales Punkt Set

Das fünf-dimensionale Punkt Set besteht, wie der Name nahelegt, aus einem fünf-dimensionalen Vektor. Jede Dimension enthält eine Eigenschaft der Musik. Der erste Wert stellt eine der wichtigsten Eigenschaften der Musik dar. Dieser Wert ist die Zeit. Auf welche Weise die Zeit im ersten Wert dargestellt wird, ist frei wählbar. Die Darstellung kann in Millisekunden oder auch als Notenummer im Track geschehen. Der zweite Wert ist die Höhe der gespielten Note. Die Höhe der Note befindet sich im Intervall (21, 108). Mit diesem Intervall können alle Noten eines Klaviers abgebildet werden. Das dritte Element ist die diatonische Notenummer. Die diatonische Tonleiter besteht aus den 7 Grundtönen einer Oktave (C,D,E,F,G,A,B,C). (Dommer, 1865)

Element vier beinhaltet, wie lange der Ton gehalten wird und Element fünf zu welcher Stimme dieser Ton gehört. (Meredith, 2006)

7.3.2. SIA

SIA steht für Structure Induction Algorithmus. Dieser Algorithmus benutzt das im vorherigen Kapitel beschriebene fünf-dimensionale Punkt Set. Da für die Zwecke der Patternerkennung in dieser Arbeit lediglich drei Punkte benötigt werden, kann der diatonische Wert und der Wert, wie lange die Note gehalten wird, verworfen werden. Diese zwei Werte können aber für andere genauere Untersuchungen, welche keinen Teil dieser Abschlussarbeit darstellen, wichtiger Bestandteil sein.

Das Tripel (Zeit, Tonhöhe, Stimme) wird daraufhin so implementiert, dass das Tupel (Zeit, Tonhöhe) Koordinaten darstellen und die Stimme die Arraynummer ist. Somit hat das Tripel (2, 64, 2) die Koordinaten (2, 64) in Array Nr. 2. Die Arrays können auch als Koordinatensystem dargestellt werden wie in Abbildung 17 zu sehen ist.

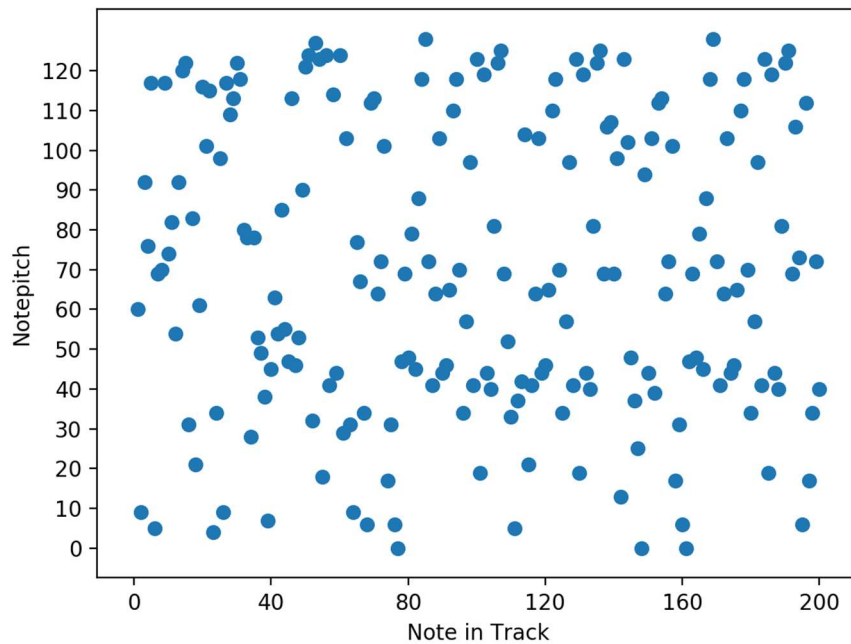


Abbildung 17: Darstellung des SIA Datensatzes für einen Track

Der nächste Schritt besteht darin, Vektoren zwischen den Punkten zu berechnen. Die Vektoren werden berechnet, da gleiche Pattern oder Keychanges dieselben Vektorenstrukturen aufweisen wie Abbildung 18 verdeutlicht. Lediglich die vertikale Spiegelung ist so nicht aufzufinden, da die Vektoren in dem Pattern kleiner oder größer werden.

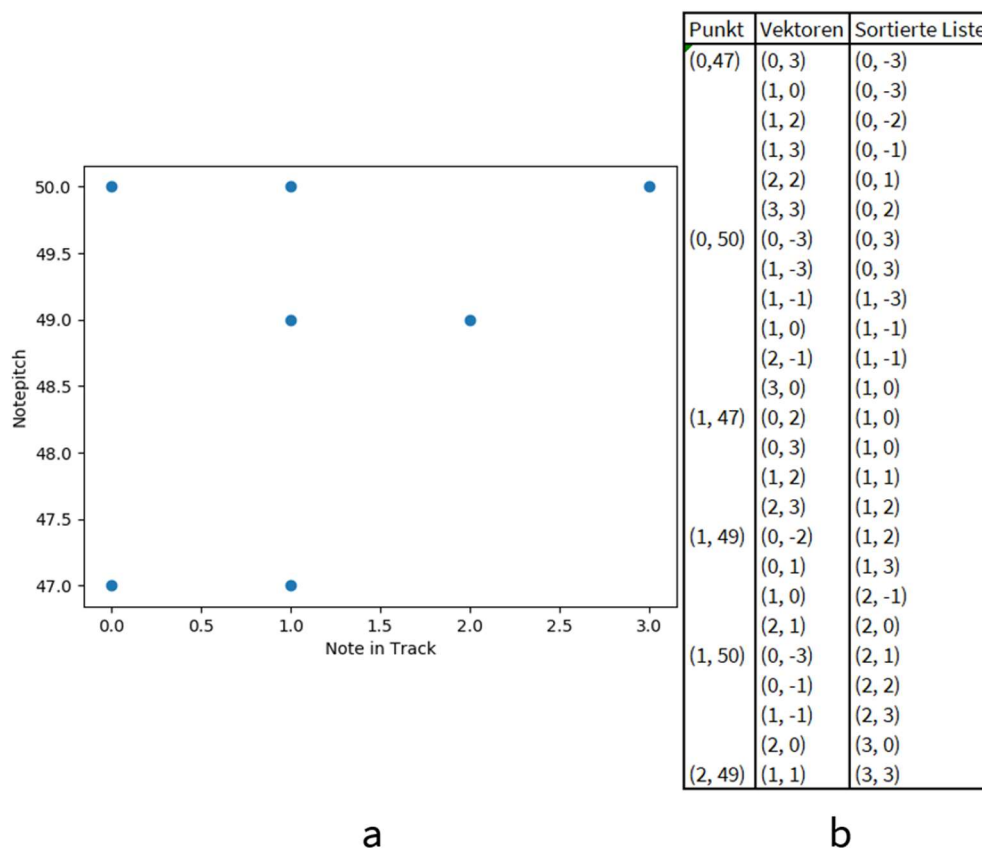


Abbildung 18: a) Noten dargestellt in Koordinatensystem

b) Tabelle erstellt aus den Punkten im Koordinatensystem. Spalte zwei beinhaltet Vektoren zwischen allen Punkten (ausgenommen sind Vektoren, in welchen der Zeitwert negativ ist) und Spalte drei die lexikografisch sortierte Liste der Vektoren

Nachdem alle Vektoren berechnet sind, müssen diese lexikografisch sortiert werden. Die Sortierung kann dabei mit einem beliebigen Sortieralgorithmus durchgeführt werden. Nach der Sortierung der Vektoren sind alle Pattern am Wert eines lexikografischen Eintrages ablesbar. Jedoch können einige Werte in einem Eintrag nicht zu dem Pattern gehören, da diese zufällig in dem Musikstück vorkommen können. Umso länger ein Musikstück ist desto mehr solcher Einträge kommen in einem Eintrag vor.

7.4. Umsetzung der Vektorgeometrischen Patternsuche

Die Vektorgeometrische Patternsuche wurde in Python umgesetzt. Zum Einlesen der MIDI-Noten wurde kein Package verwendet, da diese keine Zeitangaben auslesen konnten. Somit müssen die Lieder per Hand in die geforderte Form gebracht werden. Da für die Umsetzung kein fünf-dimensionales Punkt Set benötigt wird, wurde zur Zeitersparnis lediglich ein Tripel zum Einlesen in die erforderliche .txt-Datei benutzt. Das Tripel besteht aus (Notennr, Tonhöhe, Stimme/Tracknr.). Die Textdatei muss so aufgebaut sein, dass die Noten nach

aufsteigender Notenummer in die Datei eingetragen werden. Es ist dabei unerheblich, ob die einzelnen Tracks vermischt oder nacheinander eingetragen werden.

Nachdem die Tripel eingelesen und die dazugehörigen Vektoren berechnet sind, müssen die Vektoren sortiert werden. Dafür wurde der Sortieralgorithmus *Quick-Sort* gewählt. Zugunsten von Quick-Sort wurde sich entschieden, da die Implementierung trivial ist und er eine Laufzeit von $\theta(n * \log(n))$ im Average-Case aufweist, was eine kurze Laufzeit für Sortieralgorithmen darstellt. (Weicker, 2015) Wenn die Sortierung abgeschlossen ist, werden alle Ergebnisse in eine Textdatei eingetragen, sofern diese mehr als drei Einträge beinhalten. Die Ergebnisse zur Laufzeit und zur Genauigkeit sind in Kapitel 9 vorzufinden.

8. Patternerkennung mittels SIA und der anschließenden Auswertung durch ein CNN mit YOLOv4

Der letzte vorzustellende Methode Pattern zu erkennen, ist eine Kombination aus vektorgeometrischer Erkennung und einer Erkennung mittels eines CNN.

Die Kernidee hierbei besteht darin, dass die Koordinatensysteme von SIA abgespeichert werden und diese an ein CNN weitergegeben werden. Pattern können zwar mit bloßem Auge erkannt werden, jedoch sind kurze Pattern leicht zu übersehen. Aus diesem Grund wird ein CNN verwendet. Es soll Pattern automatisch erkennen und mit Bounding Boxes klassifizieren. Die zu kategorisierenden Klassen umfassen Pattern der Länge <5, bis 10, bis 15, bis 20, bis 25, bis 30, bis 35, bis 40, bis 45, bis 50, >50.

Für die Umsetzung wird dasselbe Programm wie für SIA verwendet. Jedoch wird die Berechnung der Abstandsvektoren und die lexikografische Sortierung der Vektoren verworfen, sodass nur das Koordinatensystem ausgegeben wird. Als Eingabe wird abermals das Tripel aus Kapitel 7.4. verwendet. Die Werte werden daraufhin in Arrays geschrieben und per Numpy in ein Koordinatensystem eingetragen und gespeichert. Numpy erzeugt im folgenden Koordinatensysteme wie jenes aus Abbildung 17. Anschließend wird ein CNN mit YOLOv4 darauf trainiert, Pattern in den Bildern zu erkennen. Die erkannten Pattern sollen dann aufgrund der enthaltenen Notenpunkte in einer Bounding Box in eine der oben genannten Gruppen eingeordnet werden. Für das Training des CNN wurden 500 Koordinatensysteme zufällig generiert, die Bilder gespeichert und die einzelnen Pattern mit *YOLO Label* kategorisiert.

Als nächstes wurden 400 der 500 Bilder zum Training verwendet und die letzten 100 Bilder zum Validieren genutzt. Das Training erfolgte analog zu dem erstellten CNN aus Kapitel 6.4..

Wie gut oder schlecht diese Methode im Gegensatz zu den anderen Methoden funktioniert, kann im nächsten Kapitel nachgelesen werden.

9. Vergleich der unterschiedlichen Verfahren der Patternerkennung

Nachdem in den letzten drei Kapiteln die einzelnen Verfahren vorgestellt wurden, werden diese im letzten Teil der Abschlussarbeit miteinander verglichen. Der Vergleich beinhaltet dabei folgende drei Aspekte:

- Rechenzeit
- Anwendbarkeit hinsichtlich des Musikgenres
- Anzahl der gefundenen Pattern

Jedoch werden nur die Verfahren ohne CNN mit YOLO v4 verglichen, da die neuronalen Netze entweder nicht trainierbar waren oder aber die Ergebnisse der Klassifizierung bei weitem nicht den Anforderungen entsprachen.

9.1. Probleme der neuronalen Netze

9.1.1 Patternerkennung mit CNN und Spektrum

Das CNN, welches mit den Spektren der Musik arbeitet, war nicht trainierbar wie unten stehender Konsolenoutput beweist.

```
1389: -nan, -nan avg loss, 0.001000 rate, 0.923646 seconds, 5556 Images, 8.288765 hours left
Loaded: 0.000044 seconds
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000000, .SR: 0.000000, .75SR: 0.000000, count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000)
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.000000, GIOU: 0.000000), Class: nan, Obj: nan, No Obj: nan, .SR: 0.000000, .75SR: 0.000000, count: 19, class_loss = -nan, iou_loss = -nan, total_loss = -nan)
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.000000, GIOU: 0.000000), Class: nan, Obj: nan, No Obj: nan, .SR: 0.000000, .75SR: 0.000000, count: 18, class_loss = -nan, iou_loss = -nan, total_loss = -nan)
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000000, .SR: 0.000000, .75SR: 0.000000, count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000)
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.000000, GIOU: 0.000000), Class: nan, Obj: nan, No Obj: nan, .SR: 0.000000, .75SR: 0.000000, count: 36, class_loss = -nan, iou_loss = -nan, total_loss = -nan)
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.000000, GIOU: 0.000000), Class: nan, Obj: nan, No Obj: nan, .SR: 0.000000, .75SR: 0.000000, count: 16, class_loss = -nan, iou_loss = -nan, total_loss = -nan)
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: -0.395610), Class: 0.000000, Obj: 0.000000, No Obj: 0.000000, .SR: 0.000000, .75SR: 0.000000, count: 1, class_loss = 2.000000, iou_loss = 0.000000, total_loss = 2.000000)
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.000000, GIOU: 0.000000), Class: nan, Obj: nan, No Obj: nan, .SR: 0.000000, .75SR: 0.000000, count: 19, class_loss = -nan, iou_loss = -nan, total_loss = -nan)
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.000000, GIOU: 0.000000), Class: nan, Obj: nan, No Obj: nan, .SR: 0.000000, .75SR: 0.000000, count: 13, class_loss = -nan, iou_loss = -nan, total_loss = -nan)
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000000, .SR: 0.000000, .75SR: 0.000000, count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000)
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.000000, GIOU: 0.000000), Class: nan, Obj: nan, No Obj: nan, .SR: 0.000000, .75SR: 0.000000, count: 9, class_loss = -nan, iou_loss = -nan, total_loss = -nan)
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.000000, GIOU: 0.000000), Class: nan, Obj: nan, No Obj: nan, .SR: 0.000000, .75SR: 0.000000, count: 14, class_loss = -nan, iou_loss = -nan, total_loss = -nan)
```

Abbildung 19: Output von darknets Trainingsinformationen

In den Outputs ist zu erkennen, dass die Einträge *Obj*, *No Obj* und *avg loss* nach circa 1300 Iterationen das Ergebnis *nan* zurückgeben. Wenn *nan* partiell auftritt, würde dieser Sachverhalt kein Problem darstellen. Da aber jeder Eintrag mit *nan* versehen ist, ist das ein Hinweis darauf, dass beim Lernen Fehler im Netz aufgetreten sind. Fehler können unter anderem ein Fehler in der Configdatei oder schlechte Trainingsdaten sein. (Bochkovskiy, Github, 2020) Um die erste Fehlerquelle zu eliminieren, wurde die Configdateien mehrmals und streng nach Vorgabe neu angelegt. Da sich durch diese versuchte Fehlerelimination keine Änderung erzeugen ließ, befindet sich der Fehler in den Trainingdaten.

Bei dem Fehler handelt es sich wahrscheinlich um die Größe der Bilder der Spektren. Das Format eines exportierten Spektrums der Trainingsdaten befindet sich, abhängig von der Songlänge, zwischen 6000x880 Pixel bis hin zu 9000x880 Pixel. Jedoch konnten das Netz nur eine Dimension von 1024x320 Pixeln annehmen. Das CNN war aufgrund der verwendeten Geforce GTX 1660 Super und dem vorhandenen Grafikkartenspeicher von 6 GB auf diese Größe beschränkt. Dem Ersteller dieser Arbeit stand auch keine Erweiterung des Speicherplatzes durch weitere Grafikkarten zur Verfügung, um zu testen, ob darin der Fehler bestand. Per CPU konnte ebenfalls nicht gerechnet werden, da für die Klassifikation der Spektren in 20 unterschiedliche Gruppen 40000 Lerniterationen vorgesehen werden müssen. (Bochkovskiy, Github, 2020) Ein AMD Ryzen CPU benötigt für 900 Lerniterationen 18 Stunden. Dadurch würde das Trainieren des Netzes circa einen Monat in Anspruch nehmen. Somit muss aus zeitlichen Gründen von einem Training mit CPU abgesehen werden.

Aufgrund der oben beschriebenen Skalierung sind Informationen des Spektrums verloren gegangen. Durch die verlorenen Informationen ist es möglich, dass das Netz keine Pattern mehr erkennen konnte und durch das Training in einen Zustand geraten ist, in welchem die Neuronen gesättigt waren. Durch die Sättigung haben die Neuronen somit nicht mehr gelernt, wodurch der *avg loss* und weitere Eigenschaften nicht mehr berechenbar war.

9.1.2. Patternerkennung mit SIA und CNN

Das CNN, welches mit den Bilddaten von SIA arbeitete, konnte erfolgreich trainiert werden, wie folgende Abbildung aufzeigt.

```
888: 3.469346, 3.332901 avg loss, 0.000622 rate, 2.705764 seconds, 14208 Images, 16.907187 hours left
Loaded: 0.000026 seconds
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIoU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000087, .SR: 0.000000, .75R: 0.000000, count: 1, class_loss = 0.0
02333, iou_loss = 0.000000, total_loss = 0.002333
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.569546, GIoU: 0.567910), Class: 0.079500, Obj: 0.022268, No Obj: 0.000875, .SR: 1.000000, .75R: 0.000000, count: 4, class_loss = 8.2
22820, iou_loss = 9.155201, total_loss = 17.378021
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.000000, GIoU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.001201, .SR: 0.000000, .75R: 0.000000, count: 1, class_loss = 0.0
93933, iou_loss = 0.000000, total_loss = 0.093933
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.513086, GIoU: 0.408520), Class: 0.984530, Obj: 0.001035, No Obj: 0.000109, .SR: 0.500000, .75R: 0.000000, count: 2, class_loss = 2.0
02329, iou_loss = 0.005035, total_loss = 0.007364
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.621300, GIoU: 0.592381), Class: 0.678885, Obj: 0.013466, No Obj: 0.001538, .SR: 0.333333, .75R: 0.333333, count: 3, class_loss = 3.9
15877, iou_loss = 11.005980, total_loss = 14.921057
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.508774, GIoU: 0.556310), Class: 0.193809, Obj: 0.012025, No Obj: 0.001225, .SR: 0.500000, .75R: 0.000000, count: 2, class_loss = 3.4
55386, iou_loss = 0.340076, total_loss = 3.794052
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIoU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000071, .SR: 0.000000, .75R: 0.000000, count: 1, class_loss = 0.0
01418, iou_loss = 0.000000, total_loss = 0.001418
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.568976, GIoU: 0.535640), Class: 0.726395, Obj: 0.004804, No Obj: 0.001104, .SR: 0.571429, .75R: 0.142857, count: 7, class_loss = 8.4
40742, iou_loss = 19.282354, total_loss = 27.731096
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.479522, GIoU: 0.468273), Class: 0.121360, Obj: 0.033721, No Obj: 0.000595, .SR: 0.000000, .75R: 0.000000, count: 2, class_loss = 3.6
09929, iou_loss = 0.925180, total_loss = 4.535109
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIoU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000106, .SR: 0.000000, .75R: 0.000000, count: 1, class_loss = 0.0
03060, iou_loss = 0.000000, total_loss = 0.030600
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.461524, GIoU: 0.427212), Class: 0.614742, Obj: 0.012538, No Obj: 0.001090, .SR: 0.500000, .75R: 0.000000, count: 6, class_loss = 7.8
23645, iou_loss = 15.059288, total_loss = 22.882933
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.630388, GIoU: 0.604576), Class: 0.133892, Obj: 0.071449, No Obj: 0.001722, .SR: 0.750000, .75R: 0.250000, count: 4, class_loss = 6.9
00636, iou_loss = 1.537259, total_loss = 0.517095
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.445388, GIoU: 0.442069), Class: 0.978314, Obj: 0.000438, No Obj: 0.000101, .SR: 0.500000, .75R: 0.000000, count: 2, class_loss = 2.0
04452, iou_loss = 10.744564, total_loss = 12.749016
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.406217, GIoU: 0.406217), Class: 0.381783, Obj: 0.026773, No Obj: 0.001108, .SR: 0.333333, .75R: 0.000000, count: 6, class_loss = 9.0
39699, iou_loss = 3.405145, total_loss = 12.444043
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.000000, GIoU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000909, .SR: 0.000000, .75R: 0.000000, count: 1, class_loss = 0.0
37182, iou_loss = 0.000000, total_loss = 0.037182
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.596611, GIoU: 0.557959), Class: 0.988792, Obj: 0.000724, No Obj: 0.000090, .SR: 1.000000, .75R: 0.000000, count: 1, class_loss = 1.0
02529, iou_loss = 0.272744, total_loss = 7.275273
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.526781, GIoU: 0.516884), Class: 0.742851, Obj: 0.003885, No Obj: 0.001053, .SR: 0.666667, .75R: 0.000000, count: 3, class_loss = 3.4
17455, iou_loss = 0.700563, total_loss = 10.190019
```

Abbildung 20: Output des Trainings nach 888 Iterationen

Dabei konnte nach Beendigung des Vorgangs ein *avg loss* von ~0,011 erzeugt werden. Jedoch waren die erzeugten Ergebnisse des Netzes nicht brauchbar. Lediglich in zwei von 7 vorgelegten Koordinatensystemen, konnten Pattern

erkannt werden. Des Weiteren passen die vom Netz gefundenen Pattern nicht zu den gelernten Klassen. Dafür gibt es zwei Gründe:

- Trainingsdaten waren zu weit entfernt von realen Daten
- YOLO v4 ist für diese Art der Patternerkennung nicht geeignet

Zum einen können vom Ersteller Fehler bei der Erstellung der Trainingsdaten entstanden sein. Fehler Nummer eins kann die fehlerhafte Unterteilung der zu klassifizierenden Klassen sein. Fehler Nummer zwei in den erzeugten Koordinatensystemen, welche später für das Training verwendet wurden. Zweiteres ist der wahrscheinlichere Fall. Die erzeugten Koordinatensysteme zum Training hatten nur circa 200 Einträge, wohingegen die Koordinatensysteme der zum Testen verwendeten Musikstücke zwischen 400-1900 Einträge besaßen. Aus diesem Grund können erlernte Eigenschaften auf die realen Daten nicht mehr zutreffen, wodurch die Pattern nicht erkannt oder falsche Pattern erkannt wurden.

Zum anderen ist es eher unüblich, dass ein YOLO v4 für eine solche Art der Patternerkennung verwendet wird. YOLO v4 versucht, für eine bestimmte Klasse markante Merkmale/Eigenschaften zu finden. Bei einer Klassifikation zwischen Hund und Katze ist eine eindeutige Klassifikation möglich, da beide Klassifikationsgruppen einzigartige Merkmale aufweisen, welche die andere Gruppe nicht besitzt. Bei Musikpattern einer bestimmten Länge können jedoch keine markanten Eigenschaften gefunden werden. Als Beispiel dient dabei folgender Sachverhalt, in welchem ein Pattern von der Länge 25 erkannt werden soll. In der klassischen Musik existieren 88 verschiedene Tonhöhen von A0 bis C8. (Drechsler, 2020) Somit existieren 88^{25} verschiedene Pattern für die Patternlänge von 25. Das sind $4,09 \cdot 10^{48}$ verschiedene Pattern. Jedes einzelne dieser Pattern besitzt verschiedene Eigenschaften.

Durch diesen Sachverhalt wird verdeutlicht, dass das Netz einer Gruppe womöglich keine eindeutigen Eigenschaften zuordnen konnte. Der bessere Ansatz für eine Art der Patternerkennung wäre es, ein Netz z.B. 15 feste Pattern lernen und diese dann in anderen Songs finden zu lassen. Wie dieser Ansatz in der Realität umgesetzt werden kann und später dann funktioniert, kann Bestandteil einer weiteren Abschlussarbeit sein, da dieser Ansatz den zeitlichen Rahmen der Abschlussarbeit übersteigt.

Aufgrund der eben beschriebenen Fehler sind folgende Klassifizierungen des CNN's getroffen worden.

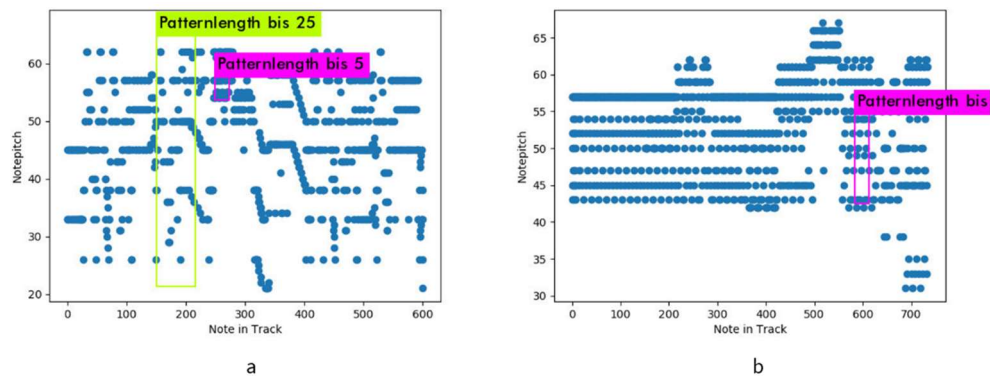


Abbildung 21: Output des CNN für den ersten MIDI-Track in Beethovens Ode an die Freude (a) und MIDI-Track zwei in Something Just Like This - Chainsmokers & Coldplay (b)

In den Klassifizierungen ist zu sehen, dass das Pattern der Länge bis 25 zu viele Noten enthält. Das gleiche tritt in Bild b bei der Patternlänge bis fünf Noten auf. Aus diesem Grund kann die Annahme getroffen werden, dass die zugeordneten Klassen nicht der zugeordneten Länge entsprechen. Außerdem werden die Pattern ausschließlich einzeln erkannt, wodurch diese streng genommen keine sich wiederholende Pattern darstellen. Damit das Netz ausschließlich Pattern erkennt, welche öfter als einmal in dem Koordinatensystem enthalten sind, bräuhete das Netz eine Art künstliche Intelligenz, welche den eben erklärten Sachverhalt in der Klassifizierung mitberücksichtigt. Dadurch fällt der zweite Ansatz mit neuronalen Netzen ebenfalls, aus Gründen der mangelnden Funktionalität, aus dem Vergleich heraus.

9.2. Rechenzeit

Die Rechenzeiten in Sekunden der einzelnen Verfahren kann Abbildung 22 entnommen werden.

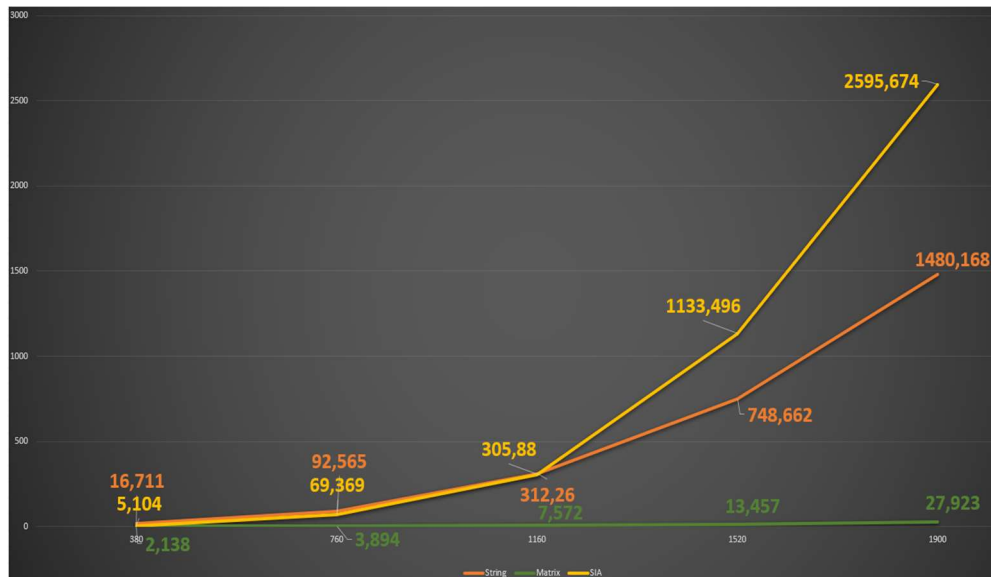


Abbildung 22: Kurvenverläufe der Patternsuche mit String (Orange), Matrix (Grün) und SIA (Gelb)

Es ist zu erkennen, dass die Variante mit der Matrix eine signifikant geringere Laufzeit aufweist als die Varianten mit SIA und String. Das liegt an dem, im Gegensatz zu den anderen beiden Algorithmen, relativ kleinen Anstieg an Rechenoperationen, die pro Note hinzukommen. Die Matrix wird zwar mit jeder weiteren Note um $Anzahl\ Noten * 2 + 1$ größer aber die untere Hälfte der Matrix wird nicht verwendet, wodurch nur $Anzahl\ Noten + 1$ mehr Werte berechnet werden müssen. Dies stellt einen nahezu quadratischen Anstieg dar. SIA muss $Anzahl\ von\ Noten$ mehr Vektoren berechnen. Der enorme Unterschied in den Rechenzeiten zwischen SIA und der Variante mit Matrix beruht auf dem Sortieren der Vektoren in SIA. Bei einer Songlänge von 1900 Noten müssen circa 1,8 Mio Vektoren lexikografisch geordnet werden, was bei der Matrixvariante hingegen komplett entfällt. Aus diesem Grund ist SIA, bei Musikstücken ab einer Länge von circa 1200 Noten langsamer als die Stringvariante.

Bei der Stringvariante müssen bei einer Tracklänge von N $\sum_{n=1}^{N-3} n + \sum_{n=1}^{N-5} n + \sum_{n=1}^{N-7} n + \dots$ Vergleiche getätigt werden. Folgende Formel ist unter folgender Voraussetzung gültig:

- Es wird ein neues Sigma nur hinzugefügt, wenn $N - ((Anzahl\ Sigma + 1) * 2 + 1) \geq 1$

Aufgrund des höheren Rechenaufwandes eines Stringvergleichs, gegenüber der Berechnung von einem Wert aus zwei Arrayinhalten, rentiert sich die Patternsuche mithilfe von Strings erst ab der Länge von 1200 Noten.

Dennoch tritt bei allen Songlängen die Matrixberechnung als Favorit in der Rechenzeit hervor.

9.3. Anzahl der gefundenen Pattern

Um die Anzahl von gefundenen Pattern in einem Musikstück auszuwerten, müssen zuerst verschiedene Musikstücke ausgewählt werden. Diese müssen dann mit den drei selbstprogrammierten Algorithmen analysiert werden. Die Songs, welche für die Analyse gewählt wurden, sind *Beethoven - Ode an die Freude*, *Darude - Sandstorm* und *Something Just Like This - Chainsmokers & Coldplay*. Die Wahl fiel auf diese drei Musikstücke, da sie mit den Genres Klassik, Pop und Elektro ein breites und diverses Spektrum der Musik abdecken. Damit soll der Beweis erbracht werden, dass die vorgestellten Algorithmen auf alle Musikrichtungen anwendbar sind.

Nachdem die drei Stücke analysiert wurden, werden die erzeugten Ergebnisdateien, mittels eines Python Programms, ausgelesen und die darin enthaltenen Pattern gezählt. Die Anzahl der gefundenen Pattern wurde daraufhin in eine separate Textdatei geschrieben, welche in Abbildung 23 zu sehen ist.

```
String Patternsearch:
Beethoven      Track1: 59
                Track2: 125
Chainsmokers    Track1: 63
                Track2: 84
Sandstorm      Track1: 15
                Track2: 52
                Track3: 71

Matrix Patternsearch:
Beethoven      Track1: 4133
                Track2: 3969
Chainsmokers    Track1: 5651
                Track2: 8635
Sandstorm      Track1: 9335
                Track2: 5214
                Track3: 27025

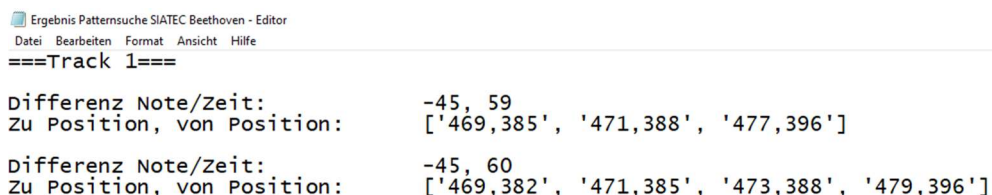
SIA Patternsearch:
Beethoven      Track1: 23624
                Track2: 19810
Chainsmokers    Track1: 17114
                Track2: 24236
Sandstorm      Track1: 8806
                Track2: 27357
                Track3: 40017
```

Abbildung 23: Ergebnisdatei mit den drei unterschiedlichen Algorithmen. Jeder Algorithmus wurde in die einzelnen zu untersuchenden Musikstücke unterteilt, welche wiederum in die Anzahl der enthaltenen MIDI-Tracks unterteilt wurden.

Vorab sei gesagt, dass an dieser Stelle nicht validiert werden kann, ob alle Pattern gefunden wurden. Um zu vergleichen, ob die computergestützten Algorithmen alle Pattern gefunden haben, wird eine manuelle Musikanalyse benötigt. In dieser stehen dann alle Pattern sauber aufgeführt. Da der Vorgang einer manuellen Musikanalyse einen sehr hohen Bedarf an Zeit und fundiertem musikalischem Wissen erfordert, wurde von einer manuellen Musikanalyse abgesehen.

Wie in oben stehender Abbildung zu sehen ist, unterscheidet sich die Anzahl der erkannten Pattern enorm. In der String basierten Patternsuche wurde nebenbei jeweils ein Array für die Position und die Länge eines gefundenen Patterns angelegt. Wenn ein neu gefundenes Pattern in den Intervall Position und Position + Länge eines bereits gefundenen Patterns fällt, so wird das neu gefundene nicht in die Ergebnisdatei übernommen. Somit zeichnet sich dieser Algorithmus durch einen geringen Grad des Post-Processing aus. In den anderen beiden Algorithmen wurde davon abgesehen, da dies einen signifikant höheren Programmieraufwand bedeutet hätte. Somit sind in den Ergebnisdateien der Matrix basierten Patternsuche alle gefundenen Pattern der Länge ≥ 3 erhalten.

In den Ergebnisdateien von SIA ist zu erkennen, dass diese eine deutlich höhere Anzahl an erkannten Pattern aufweist. Der Grund dafür liegt darin, dass alle errechneten Vektoren in ihre lexikografische Reihenfolge gebracht wurden. Jeder Eintrag der lexikografischen Ordnung wurde im Folgenden in die Ergebnisdatei übernommen, sofern dieser mehr als drei Vektoren beinhaltet. Der Eintrag der lexikografischen Ordnung wird selbst dann übernommen, wenn darin kein Pattern enthalten ist. Dieses Problem ist in Abbildung 24 zu sehen. Um die Ergebnisdatei einzukürzen und einzig die relevanten Ergebnisse anzuzeigen, muss ein aufwendiges Post-Processing durchgeführt werden. Ein Post-Processing in SIA muss zuerst einen gesamten lexikografischen Eintrag auf eine Zahlenfolge, wie zum Beispiel ['469,385', '470,386', '471,387'], durchsuchen. Eine solche Zahlenfolge stellt ein Pattern der Länge drei dar. Beim Durchsuchen der gesamten Datei müssen dann alle Einträge, welche kein Pattern enthalten, gelöscht werden, was die Ergebnisanzahl enorm senken wird. Jedoch würde das Durchsuchen der gesamten Ergebnisdatei die ohnehin schon schlechte Rechenzeit weiter erhöhen. Dadurch liegt die Anzahl der vermeidlich erkannten Pattern um ein vielfaches höher als bei der String- oder Matrixvariante. Die Pattern müssen somit manuell durchgeschaut und validiert werden.



```

Ergebnis Patternsuche SIA TEC Beethoven - Editor
Datei Bearbeiten Format Ansicht Hilfe
===Track 1===

Differenz Note/Zeit:      -45, 59
Zu Position, von Position: ['469,385', '471,388', '477,396']

Differenz Note/Zeit:      -45, 60
Zu Position, von Position: ['469,382', '471,385', '473,388', '479,396']

```

Abbildung 24: Outputs in der Ergebnisdatei von SIA TEC, in welcher keine Pattern enthalten sind.

Abschließend kann gesagt werden, dass die brauchbarsten Ergebnisse von der String-basierte Patternsuche erstellt werden, da diese ohne aufwendiges Post-Processing einen Großteil der vorkommenden Pattern abbildet und keine Doppelungen in den Ergebnissen auftreten. An Position zwei steht die Matrix-basierte Patternsuche, da in der Ergebnisdatei ohne Post-Processing alle gefundenen Pattern enthalten sind. Dennoch kommen viele Teilpatterns welche, vergleichbar zu Abbildung 25 sind, mehrfach vor. Sollte jedoch ein Post-Processing für die Matrixvariante umgesetzt werden, können die Ergebnisse auf ein Level mit der Stringvariante stehen.

Gefundene Pattern: A2 A3 E3 A2 A3 E3 A2 A3 E3 A2 A3 E3 A2, A2 A3 E3 A2 A3 E3 A2 A3 E3 A2 A3 E3 A2
 Noten im Lied: 761, 778
 Länge des Patterns: 16

Gefundene Pattern: A2 A3 E3 A2 A3 E3 A2 A3 E3 A2, A2 A3 E3 A2 A3 E3 A2 A3 E3 A2
 Noten im Lied: 764, 778
 Länge des Patterns: 13

Abbildung 25: Pattern wird einmal als normales Pattern und ein weiteres Mal als kürzeres Teilpattern erkannt.

Am schlechtesten ist in diesem Bereich SIA. Die Unmenge an unnützen Pattern des Algorithmus können nur durch ein hohes Maß an Post-Processing brauchbar gemacht werden. Ohne Post-Processing bietet SIA somit keinen großen Mehrwert.

10. Fazit

Die Technik der Convolutional Neuronal Networks, womit am Anfang die besten Ergebnisse erhofft wurden, haben auf ganzer Linie in diesem Bereich der Patternerkennung versagt. Für die Patternerkennung mit Spektren sind mindestens 24GB Grafikkartenspeicher vonnöten, um eine Aussage treffen zu könne, ob dieses Verfahren weiteres Potential besitzt oder ähnlich schlechte Ergebnisse wie die Patternerkennung mit CNN und SIA hervorbringt. Somit kann diese Art der Patternerkennung eine weitere Abschlussarbeit darstellen, sofern ein ausreichend großes Sortiment an Grafikkarten vorhanden ist oder die Technik soweit fortgeschritten ist, das 24GB Grafikkartenspeicher in einer Grafikkarte möglich werden.

Eine weitere Möglichkeit, Pattern zu finden, bietet sich mit einem rekurrenten neuronalen Netz (RNN). RNN's können zeitlich codierte Informationen aus dem Musikstück auslesen. (Beck & Rey, 2018) Dem Netz sollte dabei das Musikstück als MIDI-Stream übergeben werden. Hierbei kann dies auf drei verschiedenen Weisen geschehen. Erstens jede Zahl einzeln in das Netz schicken, zweitens das gesamte Musikstück auf einmal in das Netz geben oder drittens den MIDI-Stream mit zeitlichen Abhängigkeiten in das Netz geben. In Option drei würden somit zeitliche Abhängigkeiten erhalten bleiben, da Noten, die zusammen gespielt werden, gleichzeitig in das Netz gegeben werden. Da in dieser Arbeit aus

Zeitgründen lediglich CNN's beleuchtet wurden, kann die Patternerkennung mithilfe von RNN's Inhalt einer weiteren Abschlussarbeit sein.

Die Patternerkennung mit CNN und SIA hat ebenfalls nicht so funktioniert wie erhofft. Es wurden nicht einmal ansatzweise sich wiederholende Pattern gefunden. Nichtsdestoweniger kann man dieses Verfahren nicht als unbrauchbar einstufen, da die Anwendung des NN nicht dem klassischen Wiedererkennen von Mustern entspricht. Ein zu untersuchendes Einsatzgebiet kann die Erkennung von Plagiaten darstellen. In diesem Einsatzgebiet lernt das CNN markante Pattern eines Musikstücks und versucht, diese in anderen Musikstücken wieder zu finden.

Die programmierten Algorithmen haben die besten Ergebnisse der vorgestellten Varianten erzielt und konnten in allen Musikgenres eingesetzt werden. Dabei befinden sich die String basierte Patternsuche auf einer Höhe mit der Matrix basierten Patternsuche ohne Post-Processing. Mit einem geringen Maß an Post-Processing ist die Matrixvariante in der Rechenzeit unschlagbar und in der Ausgabe vergleichbar gut wie die Stringvariante. Einzig SIA hat nicht die erwarteten Ergebnisse erbracht. Die Rechenzeit ist zwar bis circa 1200 Noten schneller als die Stringvariante, jedoch vergrößert sich die Laufzeit mit zunehmender Länge des Musikstücks enorm. Außerdem kann mit diesem Verfahren nicht nach vertikalen Spiegelungen gesucht werden, was mit den zwei anderen Algorithmen kein Problem darstellt. Des Weiteren wird die Ausgabe nur durch ein hohes Maß an Post-Processing brauchbar. Ohne Post-Processing hat dieser Algorithmus wenig bis gar keinen Mehrwert.

Abbildungsverzeichnis

| | |
|--|----|
| Abbildung 1: Originalpattern (links) mit dem dazugehörigen zu erkennenden Pattern einer Wiederholung (rechts)..... | 6 |
| Abbildung 2: Originalpattern (links) mit einem Keychange um 4 Halbtöne nach oben (mitte) und einer Oktave nach unten (rechts) | 7 |
| Abbildung 3: Originalpattern (links) mit einer vertikalen melodischen Inversion an Note C5 (mitte) und E4 (rechts) | 7 |
| Abbildung 4: Aufzeigen des Leakage-Effektes mit einer Funktion von 2 Hz (oben) und 2.5 Hz (unten) | 10 |
| Abbildung 5: Anwenden der Hanning-Fensterfunktion auf eine Schwingung von 2.5 Hz. Die rote Linie stellt die Ausgangsfunktion dar. Blau stellt die Hanning-Funktion dar und die grüne Linie ist die berechnete Funktion. | 10 |
| Abbildung 6: Spektrum des Liedes NIVIRIO – Flashes a) Spektrum mit 32 Bins berechnet b) Spektrum mit 32768 Bins berechnet (herangezoomt) | 11 |
| Abbildung 7: a) vollständig vermaschtes Netzwerk b) Convolutional Neural Network mit einer Convolutional-Schicht einer Pooling Schicht und einem vollständig vermaschten Netzwerk Quelle: Abbildungen wurden mittels folgenden Online-Tool erstellt: http://alexlenail.me/NN-SVG/index.html | 13 |
| Abbildung 8: Anwendung der unterschiedlichen Pooling Strategien mit einem 2x2 Filterkernel | 15 |
| Abbildung 9: a) Sigmoid Aktivierungsfunktion b) Tanh Aktivierungsfunktion c) ReLU Aktivierungsfunktion | 17 |
| Abbildung 10: Erste Ableitung der Sigmoid Aktivierungsfunktion..... | 19 |
| Abbildung 11: Ergebnis eines Versuches in welchem zwischen Genauigkeit und Rechenzeit abgewägt wird. Die Form gibt die Art des verwendeten Netzes an und die Farbe die Art der Merkmalsextraktion. Quelle: (Huang, et al., 2017), S. 8..... | 22 |
| Abbildung 12: Aufbau eines Faster R-CNN Quelle (Ren, He, Girshick, & Sun, 2016) nachgestellt. | 23 |
| Abbildung 13: a) Spektrum Highpass-Filter b) Spektrum Bandpass-Filter (Zoomed in) c) Spektrum Lowpass-Filter | 24 |
| Abbildung 14: Kurvenverläufe der einzelnen Filter, wobei die x-Achse die Frequenzen und die y-Achse die Dämpfung darstellen. a) Low-Pass-Filter b) Band-Pass-Filter c) High-Pass-Filter..... | 27 |
| Abbildung 15: a) Erstellen der Labels mit YoloLabel (Einzelbild siehe Anhang) ... b) Daraus resultierende Textdatei | 29 |
| Abbildung 16: Matrix a stellt die komplette Berechnung der Matrix dar. In Matrix b ist die effizientere Variante zu sehen, da die Berechnung der unteren Hälfte ausgespart werden kann. | 32 |
| Abbildung 17: Darstellung des SIA Datensatzes für einen Track..... | 34 |
| Abbildung 18: a) Noten dargestellt in Koordinatensystem b) Tabelle erstellt aus den Punkten im Koordinatensystem. Spalte zwei beinhaltet Vektoren zwischen allen Punkten (ausgenommen sind Vektoren, in welchen der Zeitwert negativ ist) und Spalte drei die lexikografisch sortierte Liste der Vektoren | 35 |

| | |
|--|----|
| Abbildung 19: Output von darknets Trainingsinformationen | 37 |
| Abbildung 20: Output des Trainings nach 888 Iterationen..... | 38 |
| Abbildung 21: Output des CNN für den ersten MIDI-Track in Beethovens Ode an die Freude (a) und MIDI-Track zwei in Something Just Like This - Chainsmokers & Coldplay (b) | 40 |
| Abbildung 22: Kurvenverläufe der Patternsuche mit String (Orange), Matrix (Grün) und SIA (Gelb)..... | 41 |
| Abbildung 23: Ergebnisdatei mit den drei unterschiedlichen Algorithmen. Jeder Algorithmus wurde in die einzelnen zu untersuchenden Musikstücke unterteilt, welche wiederum in die Anzahl der enthaltenen MIDI-Tracks unterteilt wurden. | 43 |
| Abbildung 24: Outputs in der Ergebnisdatei von SIATEC, in welcher keine Pattern enthalten sind. | 44 |
| Abbildung 25: Pattern wird einmal als normales Pattern und ein weiteres Mal als kürzeres Teilpattern erkannt. | 45 |

Formelverzeichnis

| | |
|---|----|
| Formel 1: Formel der CTFT (Weisstein, Discrete Fourier Transform, 2015) | 8 |
| Formel 2: Formel der DFT (Weisstein, Discrete Fourier Transform, 2015) | 8 |
| Formel 3: Umformulierte DFT-Formel..... | 9 |
| Formel 4: Berechnung des Betrages eines Vektors..... | 9 |
| Formel 5: Kostenfunktion | 18 |
| Formel 6: Fehlerfunktion für ein einzelnes Neuron | 18 |
| Formel 7: Fehlerfunktion in Matrixschreibweise..... | 19 |
| Formel 8: Fehler der nächsten Schicht..... | 19 |
| Formel 9: Formel zum Ändern der Gewichte..... | 20 |

Literaturverzeichnis

- ArcGIS. (2019). *How single-shot detector (SSD) works?* Retrieved Januar 31, 2020, from ArcGIS: <https://developers.arcgis.com/python/guide/how-ssd-works/>
- Beck, F., & Rey, G. (2018). *Neuronale Netze: Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung*. Deutschland: hogrefe. Retrieved Januar 17, 2020, from Neuronale Netze: <http://www.neuralesnetz.de>
- Bochkovskiy, A. (2020, April 28). *Github*. Retrieved from AlexeyAB/darknet: <https://github.com/AlexeyAB/darknet>
- Bochkovskiy, A., Wang, C.-Y., & Mark Liao, H.-Y. (2020, April 23). YOLOv4: Optimal Speed and Accuracy of Object Detection. Retrieved from <https://arxiv.org/pdf/2004.10934.pdf>

- Brownlee, D. J. (2019, Januar 9). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Retrieved Januar 17, 2020, from Machine Learning Mastery: <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>
- Brownlee, D. J. (2019, August 19). *Crash Course in Convolutional Neural Networks for Machine Learning*. Retrieved Januar 17, 2020, from Machine Learning Mastery: <https://machinelearningmastery.com/crash-course-convolutional-neural-networks/>
- Brownlee, D. J. (2019, September 26). *How Do Convolutional Layers Work in Deep Learning Neural Networks?* Retrieved Januar 16, 2020, from Machine Learning Mastery: <https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/>
- Brownlee, D. J. (2019, August 6). *How to Avoid Overfitting in Deep Learning Neural Networks*. Retrieved Januar 17, 2020, from Machine Learning Mastery: <https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/>
- Cannam, C., Landone, C., & Sandler, M. (2010). *A Brief Reference*. Retrieved Januar 2, 2020, from Sonic Visualiser: <https://www.sonicvisualiser.org/doc/reference/1.3/en/>
- Cannam, C., Landone, C., & Sandler, M. (2019, Dezember). *Visualisation, analysis, and annotation of music audio recordings*. Retrieved Februar 5, 2020, from Sonic Visualizer: <https://www.sonicvisualiser.org/>
- Dommer, A. v. (1865). *Musiklexikon: Was bedeutet Diatonisch?* Retrieved from musikwissenschaften.de: <https://musikwissenschaften.de/lexikon/d/diatonisch/>
- Drechsler, R. (2020, März 5). Klärung Musikalischer Grundlagen. (N. Mehlhose, Interviewer) Leipzig, Sachsen, Deutschland.
- EasyCalculation.com. (n.d.). BiQuadCalculator. Retrieved from <https://www.easycalculation.com/physics/electromagnetism/biquad-calculator.php>
- Gedan, J. (2006). *Akustische Grundlagen der Musik*. Deutschland. Retrieved from <http://www.pian-e-forte.de/texte/pdf/basics.pdf>
- Glorot, X., Bordes, A., & Bengio, Y. (2011). *Deep Sparse Rectifier Neural Networks*. Retrieved Januar 17, 2020, from Proceedings of Machine Learning Research: <http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf>
- Gonzales, R. R. (1996). *Theorie der neuronalen Netze*. Berlin: Springer. Retrieved Januar 17, 2020, from <https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf>

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. Retrieved from <https://www.deeplearningbook.org/>
- Grel, T. (2017, Februar 28). *Region of interest pooling explained*. Retrieved Januar 31, 2020, from deepsense.ai: <https://deepsense.ai/region-of-interest-pooling-explained/>
- Hsu, J.-L., & Chen, A. (1998). Efficient Repeating Pattern Finding in Music Databases. *International Conference on Information and Knowledge Management*, (p. 9). Bethesda, Maryland, USA. doi:10.1145/288627.288668
- Huang, J., Fathi, A., Rathod, V., Fischer, I., Sun, C., Wojna, Z., . . . Guadarrama, S. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. Retrieved Januar 31, 2020, from <https://arxiv.org/pdf/1611.10012.pdf>
- Imagenet. (2017, July 26). *Large Scale Visual Recognition Challenge 2017 (ILSVRC2017)*. Retrieved Januar 7, 2020, from Imagenet: <http://imagenet.org/challenges/LSVRC/2017/results>
- Karpathy, D. A. (2018, Oktober 17). *CS231n Convolutional Neural Networks for Visual Recognition*. Retrieved Januar 16, 2020, from Github: <http://cs231n.github.io/convolutional-networks/#add>
- Kriesel, V. (2013, Februar). Music Synchronization, Audio Matching, Pattern Detection, and User Interfaces for a Digital Music Library System. Bonn, Nordrhein-Westfalen, Deutschland. Retrieved März 18, 2020, from <http://hss.ulb.uni-bonn.de/2013/3271/3271.pdf>
- Krizhevsky, A., Sutskever, I., & Geoffrey, H. (2012). *ImageNet Classification with Deep Convolutional*. Retrieved Mai 23, 2020, from <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- Lutus, P. (2019, Oktober 26). BiQuadDesigner. Retrieved from <https://arachnoid.com/BiQuadDesigner/index.html>
- Math Works. (n.d.). *Convolutional Neural Network: Drei Dinge, die Sie wissen sollten*. Retrieved Januar 8, 2020, from Math Works: <https://de.mathworks.com/solutions/deep-learning/convolutional-neural-network.html#howitworks>
- Meredith, D. (2006). Point-set algorithms for pattern discovery and pattern matching in music. London. Retrieved März 15, 2020, from <https://drops.dagstuhl.de/opus/volltexte/2006/652/pdf/06171.MeredithDavid.Paper.652.pdf>

- MIDI Association. (2019). *MIDI Association*. Retrieved Februar 26, 2020, from Summary of MIDI Messages: <https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message>
- Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press. Retrieved Januar 17, 2020, from <http://neuralnetworksanddeeplearning.com>
- Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv. Retrieved Januar 31, 2020, from <https://pjreddie.com/media/files/papers/YOLOv3.pdf>
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016, Mai 9). You Only Look Once:. Retrieved from <https://arxiv.org/pdf/1506.02640.pdf>
- Redmon, N. (2003, Februar 28). *Biquads*. Retrieved Februar 5, 2020, from EARLevel Engineering: <https://www.earlevel.com/main/2003/02/28/biquads/>
- Redmon, N. (2013, Oktober 13). Biquad calculator v2. Retrieved from <https://www.earlevel.com/main/2013/10/13/biquad-calculator-v2/>
- Ren, S., He, K., Girshick, R., & Sun, J. (2016, Januar 6). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Retrieved Januar 31, 2020, from <https://arxiv.org/pdf/1506.01497.pdf>
- Roberts, P. S. (2017). *University of Oxford*. Retrieved Dezember 31, 2019, from Lecture 7 - The Discrete Fourier: <http://www.robots.ox.ac.uk/~sjrob/Teaching/SP/l7.pdf>
- Smith, J. O. (2007). *Mathematics of the Discrete Fourier Transform (DFT)* (2. ed.). W3K Publishing. Retrieved Mai 23, 2020
- Smith, S. W. (1997). *The Scientist & Engineer's Guide to Digital Signal Processing*. USA, Kalifornien: California Technical Pub.
- Strick, H. K. (2012, Juli 1). *Joseph Fourier (1768–1830)*. Retrieved Dezember 18, 2019, from Spektrum: <https://www.spektrum.de/wissen/joseph-fourier-1768-1830/1156113>
- Teufel. (2019, Juli 9). *Die Abtastrate – Tastend nach dem besten Sound*. Retrieved Januar 2, 2020, from Teufel: <https://blog.teufel.de/abtastrate/#chapter2>
- Thormählen, P. D. (2018, April 23). *Multimediale Signalverarbeitung Frequenztransformation*. Retrieved Dezember 19, 2019, from Philipps Universität Marburg: https://www.mathematik.uni-marburg.de/~thormae/lectures/mmk/mmk_3_2_ger_web.html#1
- Weicker, K. (2015). *Algorithmen und Datenstrukturen*. Leipzig.

Weisstein, E. W. (2005, April 15). *Nyquist Frequency*. Retrieved Dezember 19, 2019, from Wolfram Math World: view-source:<http://mathworld.wolfram.com/NyquistFrequency.html>

Weisstein, E. W. (2015, Februar 2). *Apodization Function*. Retrieved Dezember 31, 2019, from Wolfram Math World: <http://mathworld.wolfram.com/ApodizationFunction.html>

Weisstein, E. W. (2015, Februar 2). *Discrete Fourier Transform*. Retrieved Dezember 19, 2019, from Wolfram Math World: <http://mathworld.wolfram.com/DiscreteFourierTransform.html>

Zhang, A., C. Lipton, Z., Li, M., & J. Smola, A. (2020). Dive into Deep Learning. Retrieved Januar 31, 2020, from <https://d2l.ai>

Gesprächsleitfaden

Gesprächsleitfaden zur Klärung Musikalischer Grundlagen

Interview wurde durchgeführt am 05.03.2020 um 20:15 mit Raphael Drechsler, welcher sich in seiner Freizeit intensiv mit Musik beschäftigt und einen Master in Informatik besitzt.

| Gesprächsverlauf/Aktion | Antworten |
|--|--|
| Welche Pattern sollte man in elektronischer Musik erkennen? (Meine Patternideen gesagt) | <ul style="list-style-type: none"> - Keychange (Pattern wird eine bestimmte Anzahl von Oktaven oder Noten höher/tiefer gespielt als erstes Aufkommen des Pattern) - Spiegelung (dabei sollte sich eher auf die horizontale als die vertikale Spiegelung konzentriert werden (melodic inversion)) |
| Wieso keine horizontale Spiegelung? | - Seiner Meinung nach eher unüblich in der Musik |
| Welche weiteren Pattern sollte man Erkennen? | <ul style="list-style-type: none"> - Erklärung was Quinte ist - Quintenzirkel erklärt und gesagt das meistens auch 3 Dur und 1 Moll oder 3 Moll und 1 Dur verwendet wird -> Erkennung von Dur und Moll - Erkennung des „Millenial Woop“ (keine Kenntnis über genauen Aufbau gehabt -> muss sich selbst angelemt werden) |
| Welche Probleme kann es bei der Erkennung geben? | <ul style="list-style-type: none"> - Akkorde können neben Grundton, Terz und Quinte aus weiteren Noten zusammengesetzt oder alteriert sein. Akkord muss nicht zwingend als solcher gespielt sein. Aus Melodiespuren lassen sich nicht zwingend die verwendeten Akkorde ableiten. - Hoher Rechenaufwand |
| Vorzeigen des „KORG volca fm“ Synthesizer | <ul style="list-style-type: none"> - Vorzeigen und erklären eines Arpeggio (aufgelöster Akkord) - Arpeggios sind in der elektronischen Musik von hoher Bedeutung, da so Akkorde dargestellt werden -> nicht von weiteren Interesse für diese Masterarbeit, da diese nie gleichzeitig ausgewertet werden weil die Noten nur sequenziell und nicht parallel dargestellt werden (im String-Basierten und Matrix-Basierten Teil der Arbeit) |
| Vorzeigen der gezeigten Spektren | <ul style="list-style-type: none"> - Könnte eine schwierige Aufgabe werden, da die Spektren der einzelnen Instrumente sich vermischen. - Gut erkennlicher Patternverlauf in High- und Low-Pass-Spektren |

Bemerkung: Es wurden einige Beispiele an einem Klavier oder Synthesizer vorgespielt. Diese werden nicht mit in dem Gesprächsleitfaden mit aufgeführt da diese durchgehend enthalten sind.

Hiermit bestätige ich (Raphael Drechsler), dass der oben beschriebene Gesprächsablauf auf diese Art und Weise abgelaufen ist.

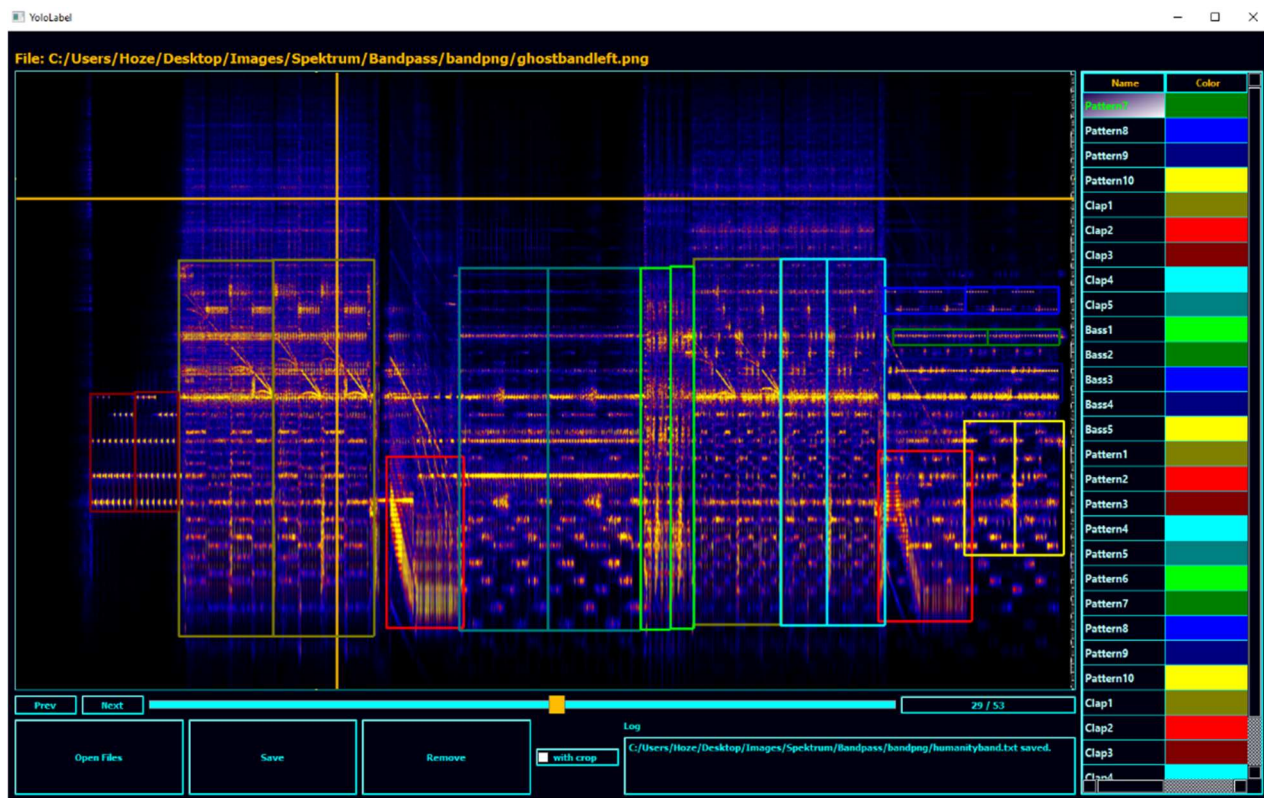
17.05.20, Leipzig

Datum, Ort

Raphael Drechsler

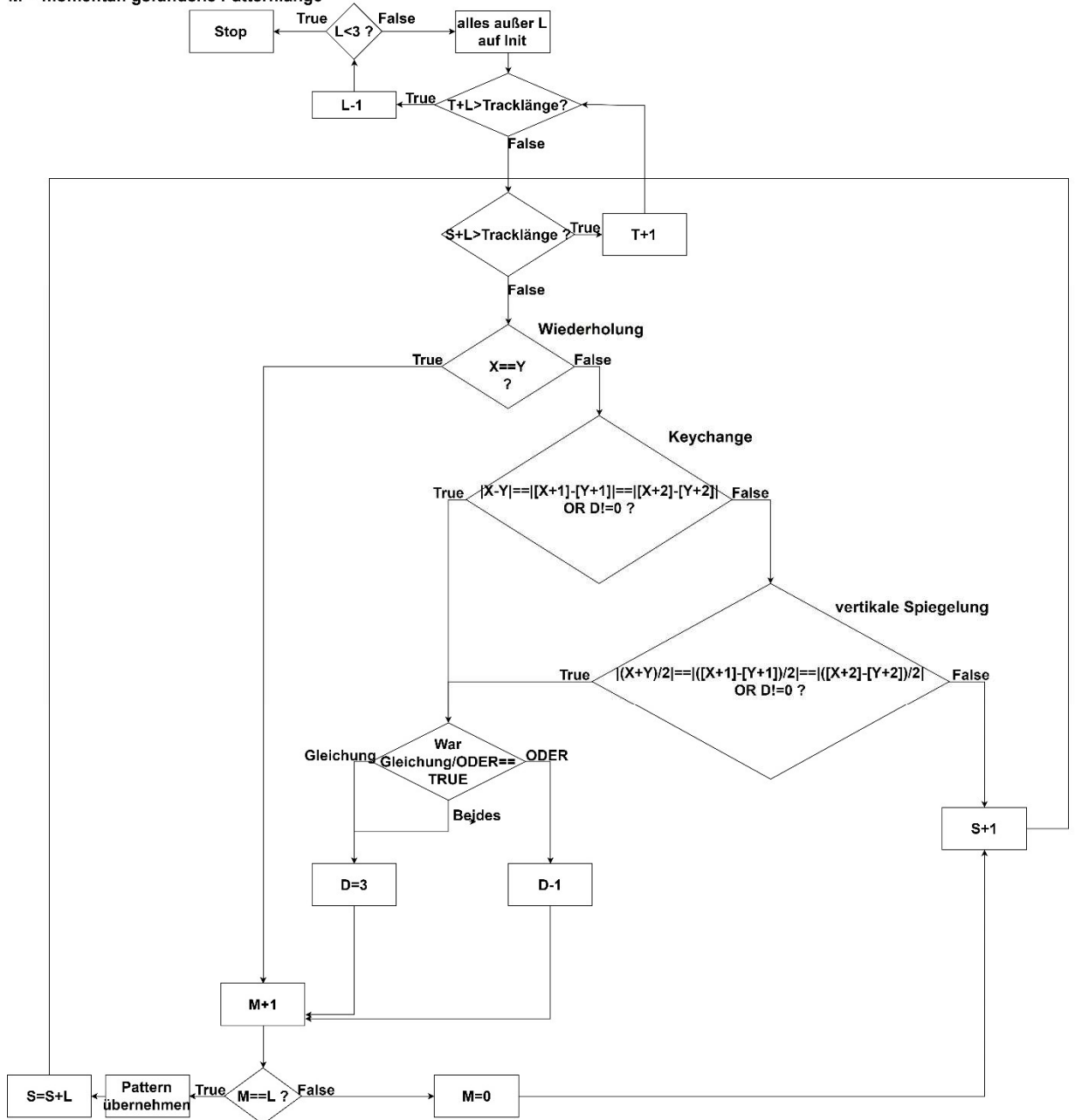
Unterschrift

Anhang



Anhang 1: Bild des Labelprozesses (Größere Version)

X = Position in Pattern
T = Trackposition des Patterns
Y = Position in möglicher Wiederholung
S = Startposition der möglichen Wiederholung
L = Länge des zu findenden Patterns
D = Keychange oder Spiegelung gefunden?
M = momentan gefundene Patternlänge



Anhang 2: Prozessablaufdiagramm der String basierten Patternsuche. Eckige Klammern beziehen sich auf Positionen und runde Klammern auf MIDI-Zahlenwerte. Initialzustände sind: $X=0$, $T=0$, $Y=0$, $S=T+L$, $L=\text{Tracklänge}/2$ (abgerundet), $D=0$, $M=0$. Ein Keychange ist erkannt, wenn die drei zu vergleichenden Positionen denselben Abstand zwischen den Noten aufweisen. Eine vertikale Spiegelung ist erkannt, wenn das Mittel aller zu vergleichenden Positionen übereinstimmen.

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich diese Masterarbeit selbstständig ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Alle den benutzten Quellen wörtlich oder sinngemäß entnommenen Stellen sind als solche einzeln kenntlich gemacht.

Diese Arbeit ist bislang keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht worden.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Leipzig, den