Inhaltsverzeichnis

[Abstract 4](#_Toc39246062)

[Danksagung 4](#_Toc39246063)

[1. Einleitung 4](#_Toc39246064)

[2. Arten von Pattern 4](#_Toc39246065)

[3. Die Fourier-Analyse 5](#_Toc39246066)

[3.1. Die diskrete Fouriertransformation 6](#_Toc39246067)

[4. Die Spektralanalyse 8](#_Toc39246068)

[5. Convolutional Neural Network 9](#_Toc39246069)

[5.1. Funktionsweise der einzelnen Bestandteile 10](#_Toc39246070)

[5.1.1. Convolutional Schicht 10](#_Toc39246071)

[5.1.2. Pooling Schicht 11](#_Toc39246072)

[5.1.3. Vollständig Vermaschtes Netzwerk 12](#_Toc39246073)

[5.2. Die ReLU-Aktivierungsfunktionen 13](#_Toc39246074)

[5.3. Backpropagation (vllt Seitenumbruch erzwingen) 14](#_Toc39246075)

[5.4. Segmentation Algorithmen 17](#_Toc39246076)

[5.4.1. You only look once (YOLO) 17](#_Toc39246077)

[5.4.2. Single Shoot Detector (SSD) 18](#_Toc39246078)

[5.4.3. Faster Region-Based Convolutional Neuronal Network (Faster R-CNN) 19](#_Toc39246079)

[6. Realisierung der Pattern Erkennung mittels einem Convolutional Neuronal Network 21](#_Toc39246080)

[6.1. Realisierung des Low-/Band-/High-Pass-Filter 22](#_Toc39246081)

[6.2. Erzeugen der Spektren 24](#_Toc39246082)

[6.3. Labeln der Erzeugten Spektren 24](#_Toc39246083)

[6.4. Erstellen des Convolutional Neuronal Networks 25](#_Toc39246084)

[7. Pattern Erkennung mithilfe von Rhythmus und Melodie 25](#_Toc39246085)

[7.1. String basierte Patternsuche 26](#_Toc39246086)

[7.2. Pattern Erkennung mithilfe von Matrizen 27](#_Toc39246087)

[7.3. Vektorgeometrische Pattern suche 28](#_Toc39246088)

[7.3.1. Fünf-Dimensionales Punkt Set 28](#_Toc39246089)

[7.3.1.1. SIA 28](#_Toc39246090)

[7.3.1.2. SIATEC 28](#_Toc39246091)

[7.4. Umsetzung der Vektorgeometrischen Pattern suche 28](#_Toc39246092)

[8. Pattern Erkennung mittels SIA und der anschließenden Auswertung durch ein CNN mit YOLOv4 28](#_Toc39246093)

[9. Vergleich der unterschiedlichen Verfahren der Pattern Erkennung 29](#_Toc39246094)

[9.1. Anwendungsbereiche hinsichtlich Musikgenre 29](#_Toc39246095)

[9.2. Rechenzeit 29](#_Toc39246096)

[9.3. Anzahl der gefundenen Pattern 29](#_Toc39246097)

[10. Fazit 29](#_Toc39246098)

[Abbildungsverzeichnis 29](#_Toc39246099)

[Tabellenverzeichnis 29](#_Toc39246100)

[Literaturverzeichnis 29](#_Toc39246101)

[Eidesstattliche Erklärung 33](#_Toc39246102)

# Abstract

# Danksagung

# 1. Einleitung

# 2. Arten von Pattern

Bevor mit den Kapiteln für die Grundlagen angefangen wird, dreht sich dieses Kapitel um die zu erkennenden Pattern. Dabei sei angemerkt das 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 kann man diese für weitere Informationen benutzten. Bei dem Keychange soll ebenfalls entschieden werden, ob der Keychange teil eines Quintenzirkels ist. Weitere Informationen werden jedoch nicht aus den Daten ausgelesen, da das musikalische Kenntnis bei dem Verfasser dieser Arbeit ab diesen Punkt nur noch mangelhaft vorzufinden sind. Somit könnte das Auslesen weiterer musikalischer Merkmale Teil einer anderen Abschlussarbeit sein.

**Wiederholung**

Die Wiederholung der gespielten Noten ist am einfachsten zu erkennende Struktur in Musikstücken. Die mindestlänge einer Wiederholung wird in dieser Arbeit bei drei Noten liegen, da drei Noten einen Akkord darstellen können. (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 verlängern. Die Maximallänge eines Patterns liegt bei Anzahl der Noten ganzzahlig geteilt durch zwei.

Abbildung 1 zeigt ein Beispiel einer Wiederholung, welche in dieser Arbeit erkannt werden soll.

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

**Keychange**

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

Untergestellte Abbildung zeigt einige Beispiele für den Keychange.

**Ein Bild, das Antenne enthält.

Automatisch generierte Beschreibung**

Abbildung : 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 wenn im Originalpattern die Noten höher wurden, werden die Noten bei der melodischen Inversion tiefer. (Drechsler, 2020), (Kriesel, 2013) Die Mindestlänge der Pattern ist die selbe wie bei dem Keychange und der Wiederholung.

Ein Bild, das Antenne enthält.

Automatisch generierte Beschreibung

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

# 3. Die Fourier-Analyse

Für jedes wissenschaftliche Feld, welches mit zum Beispiel mechanischen Schwingungen, elektrischen Schwingungen oder auch mit der Bildverarbeitung in Berührung kommt, ist die Fourier-Analyse ein wichtiges Werkzeug. Diese 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 zeitdisktrete 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, weil die DFT/DTFT diskrete Werte und eine endliche Länge besitzen (Smith, 1997).

# 3.1. Die diskrete Fouriertransformation

Wie im vorherigen Kapitel schon beschrieben, ist die diskrete Fouriertransformation die einzige Transformation, welche vom Computer berechnet werden kann. Um aber die diskrete und endliche Funktion der DFT zu verstehen, wird an dieser Stelle aufgezeigt wie sich die Formeln der DFT und der kontinuierlichen Fouriertransformation (CTFT) unterscheiden.

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

Formel : 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 bei dem Tausch des Integrals durch die Summe und bei dem Exponenten der e-Funktion. Das Integral in der DFT entfällt und wurde durch ein 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 substituiert wurde. Aus der DFT ist *n* gleichzusetzten mit dem aus der CTFT stammende *t* (Weisstein, 2015). Die DFT-Formel kann noch vereinfacht werden indem mit und die *e*-Funktion durch die Eulersche Identitätsubstituiert wird. Dadurch erhält man folgende Formel. (Thormählen, 2018)

Formel : Umformulierte DFT-Formel

Somit müssen für die *x* nur noch die einzelnen Samplewerte eingetragen und die Formel ausgerechnet 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 Punkt zur positiven x-Achse beschreibt die Phasenverschiebung der Schwingung und die Entfernung des eingetragenen Punktes zum Koordinatenursprung beschreibt die Amplitude der -ten Schwingung.

Formel : Berechnung des Betrages eines Vektors

Welche Frequenz die -te Schwingung hat, hängt von der Abtastfrequenz und der Anzahl der Abtastpunkte ab. Die Frequenzdifferenz der Punkte berechnet sich durch . (Hermann, 2010)

Bevor Formel 4 benutzt werden kann, um die Amplitude zu berechnen, müssen alle Samplewerte deren Index ≥ Anzahl Samples/2 sind gelöscht werden. Dieser Wert wird das Nyquist-Limit genannt. Es besagt das 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. (Hermann, 2010)

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

Ein Bild, das Text, Karte enthält.

Automatisch generierte Beschreibung

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

Wie in der vorgestellten Abbildung zu sehen, ist in dem Zeitfenster eine Sprungstelle zu sehen. Die Sprungstelle befindet sich an den Rändern des Ursprungsignals. Durch den daraus resultierenden ungleichmäßigen Kurvenverlauf kommt es zu einer „*Verschmierung*“ des Spektrums. Der *Leakage*-Effekt kann mit Fenstfunktionen abgeschwächt werden. Wie in **Abbildung 2** zu sehen ist, schwächen diese Funktionen die Ränder des Ursprungsignals so ab, dass diese gegen Null gehen. Dadurch werden bei einem wiederholen des Signals nur noch möglichst kleine Sprünge im Kurvenverlauf vorkommen (Roberts, 2017). Einige Fensterfunktionen sind unter anderem Barlett-, Gauß- oder die Hanning-Festerfunktion (Weisstein, 2015).

Ein Bild, das Himmel enthält.

Automatisch generierte Beschreibung

Abbildung : Anwenden der Hanning-Fensterfunktion auf eine Schwingung von 2.5 Hz

*Aliasing* tritt auf, wenn die Samplerate zu gering für hohe Frequenzen ist. Um Aliasing zu verhindern muss die Samplerate angehoben werden oder das Signal muss 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. Dabei hängt die Länge der einzelnen Fouriertransformationen und die Auflösung der Frequenzen von der Größe des betrachteten Fensters ab. Bei einer Samplerate von 44100 Samples pro Sekunde, was als eine Standartgröße bei Musik auf CDs angesehen werden kann (Teufel, 2019), bedeutet dies, dass der Abstand zwischen 2 Samples Rund 22,67 µs beträgt. Daraus lässt sich wiederum Schlussfolgern, dass bei einer Fenstergröße von 32 Samples die zeitliche Genauigkeit, durch die Fenstergröße von 0,7 ms, im Mittelpunkt steht aber die Frequenz vernachlässigt wird. Die Vernachlässigung beruht auf dem in Punkt 3.1 beschrieben Nyquist-Limit, wodurch der Frequenzbereich in 15 gleichgroße Bereiche geteilt wird. Bei 44100 Samples pro Sekunde und der daraus resultierenden maximalen Frequenz von 22050 Hz besitzt jeder Bereich ein Delta von 1470 Frequenzen, was für eine Erkennung von Pattern unbrauchbar ist, wie **Abbildung 3** 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 : 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 3.b** sind, automatisch verarbeiten zu können, werden neuronale Netze benötigt. Dafür wird ein Convolutional Neural Network (CNN) benutzt, da diese besser geeignet sind für die Klassifizierung und Bestimmung der Position von Objekten in Bildern als andere Arten von neuronalen Netzwerken. Der wichtigste Vorteil ist die Genauigkeit, mit der das CNN die geforderte Klassifizierung vornimmt. In dem Bereich der Pattern Erkennung sind CNN’s seit circa 9 Jahren besser als andere Typen von neuronalen Netzen. (Imagenet, 2017)

Der zweite Vorteil gegenüber anderen neuronalen Netzwerken ist die Effizienz. Durch die Pooling-Schichten der CNN’s 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 *mxn Poolingmatrix* übernommen. (Karpathy, 2018)

Die in diesem Kapitel vorgestellten CNN‘s haben einen unterschiedlichen Aufbau als andere 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 diese vollständig oder nur 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 2700 Pixel. Dies führt dazu das eine Vollvermaschung mit der ersten Hidden Layer 7,29 Mio Gewichte anlernen muss. Dadurch ist die Verarbeitung von Bildern nur noch bedingt effizient möglich. CNN’s 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 alleine zwischen Input Schicht und erster Hidden Layer circa 290 mal mehr Gewichte anlernen als das CNN.

Die zweite Komponente der CNN’s ist die *Convolutional* Schicht, welche Matrizen über die Pixelmatrix der Vorgängerschicht schiebt, um gewünschte Eigenschaften stärker hervor zu heben. Komponente drei ist die *Pooling* Schicht. Das Pooling bewirkt, dass gewisse Informationen, innerhalb einer mxn Matrix, verworfen werden. Dadurch wird die Anzahl der Pixel des Bildes verkleinert. Der letzte Baustein ist ein voll vermaschtes 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 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 stadtfindende Faltung (engl. convolution), werden vom Ersteller gewünschten Merkmale extrahiert. **ToDo: In höheren Schichten Einzelne Merkmale in Tieferen Schichten was anderes + Bild mit möglichen Merkmalen.** Die Merkmale werden mit Hilfe von Filterkerneln extrahiert. Der Filterkernel 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 in der dritten Dimension nicht > 1 sein. (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 Bildermatrix 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 Größe und Anzahl des verwendeten Kernels, größer. (Karpathy, 2018)

Da größere Filter jedoch 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 Bildermatrix mit einem *zero-padding* so zu vergrößern. Durch das zero-padding werde die Bilder am Rand mit 0 erweitert bis das Filter an Position 1,1 im Ausgangsbild die Berechnung durchzuführen. Durch die Erweiterung der Bildmatrix mit Nullen ist die Berechnete Feature Map nach der Faltung genauso Größe wie die Bildermatrix vor der Faltung. Die zweite Möglichkeit ist es nur valide Startpunkte für den Filter zu wählen. Bei einer 3x3 Filtermatrix würde somit die erste Berechnung an Bildpunkt 2x2 anfangen, bei einem 5x5 Bild an Position 3x3 usw. Wenn die zweite Option gewählt wird, verkleinert sich die Featuremap in Abhängigkeit der Größe des Kernels. (Karpathy, 2018)

# 5.1.2. Pooling Schicht

Die Pooling Schicht dient dazu, die eben errechneten Featuremaps zu verkleinern. Durch eine Verkleinerung der Featuremap sinkt die Anzahl der zu lernenden Parameter und der Rechenaufwand. Dadurch wird auch ein *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 Ergebnisse von neuen Klassifizierungen 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 Zahlenwerten. Max Pooling nimmt sich nur 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 : Anwendung der Unterschiedlichen Pooling Strategien mit einem 2x2 Filterkernel

In der obigen Abbildung soll die Ausgangsmatrix ein Bild mit schwarzen Rand und immer heller werdenden Inhalt darstellen. In den Ergebnismatrizen ist zu sehen, dass jede Poolingstrategie unterschiedliche Feature Maps errechnet, welche unterschiedliche Erkennungsraten zur Folge haben können.

Des Weiteren kann über das freie Festlegen der Schrittweite ein überlappendes Pooling erzeugt werden. Normalerweise wird die Schrittgröße der Matrix 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 zu dem regulären Pooling, sind ein kleinerer Informationsverlust nach dem Pooling, bessere Erkennungsraten bei gleichbleibenden Algorithmen und Struktur und der Overfitting-Effekt ist schwieriger zu erreichen. Die Nachteile des Überlappenden Poolings sind, dass die Größe der Feature Maps nicht so schnell an Größe verlieren und ein erhöhter Rechenaufwand. (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 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 Rectefied Linear Unit und steht streng genommen nicht für die Aktivierungsfunktion für sich. Es steht für den Teil des CNN’s, welches die lineare Korregierung (engl. Rectefied 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.

Aktivierungsfunktionen bilden den Netzinput auf ein Aktivitätslevel ab. Diese Abbildung geschieht häufig in einem 2-dimensionalen Diagramm in welchem die x-Achse der Netzinput 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 Aktivierungslevels des Netzes.

ReLU bringt allerdings mehrere Vorteile im Bereich des Deep Learnings, gegenüber Sigmoid und Tanh, mit sich. Der erste Vorteil ist, dass die ReLU-Funktion, keinen *vanishing gradiant effect* besitzt. Vanishing gradiant 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 wichtige Informationen darstellen. Außerdem sind die Funktionen in der Nähe der Extremwerte zu unsensibel. Deshalb sind die möglichen Veränderungen, im Gegensatz zur ReLU, nur marginal möglich. Backpropagation rechnet diese marginalen Fehler auch noch 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 nur sehr schlecht bis gar nicht. (Brownlee, 2019)

Ein Bild, das Himmel enthält.

Automatisch generierte Beschreibung

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

Wie in **Abbildung ?** zu sehen ist, ist die ReLU Funktion eine im positiven Bereich stätige 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 unter anderem in ihrem Paper heraus, das 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 (vllt Seitenumbruch erzwingen)

Folgender Text ist aus den Quellen (Gonzales, 1996), (Nielsen, 2015) zusammengestellt. Da mehrmals zwischen den Quellen gesprungen wird, was den Lesefluss unterbrechen würde, wird in diesem Kapitel auf genaue Literaturverweise verzichtet. Des Weiteren 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 die Backpropagation alle benötigt werden diese hier hintereinander aufgezählt.

Stellt das Gewicht zwischen Neuron *k* der Schicht *l-1* und Neuron *j* der Schicht *l* dar

Bias Neuron des j-ten Neurons der Schicht l

Aktivierung des j-ten Neurons der Schicht l

Gewichteter Input zu Neuron j in Schicht l

Fehler des j-ten Neurons der Schicht l

γ Schrittweite bei der Anpassung der Gewichte

б 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 Versteckten 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. Jedoch unterscheiden sich der errechnete Outputvektor mit dem gewünschten Outputvektor.

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

Formel : Kostenfunktion

*n* ist die Anzahl der verwendeten Testdaten, *x* sind ein einzelner Testdatensatz *y(x)* ist der gewünschte Outputvektor für das Trainingssample *x* und ist die Aktivierung der Schicht *L*. Da *L* die Anzahl der Schichten im Netz darstellt ist somit der momentane Output des Netzes.

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 mit der Kostenfunktion für Testsample *x* die getroffene Annahme erfüllt. Die Fehlerfunktion für Backpropagation lautet wie folgt.

Formel : Fehlerfunktion für ein einzelnes Neuron

Optional kann die Fehlerfunktion für Matrizen zur Berechnung genommen werden.

Formel : Fehlerfunktion in Matrixschreibweise

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

Formel : Fehler der nächsten Schicht

Formel 8 ist essenziell für die Backpropagation. Die Fehlerfunktion, für den Fehler der nächsten Schicht, ist jedoch gut verständlich. Wenn das NN den Backpropagation Vorgang starten, liegen dem Netz bereits folgende Informationen vor. sind die Gewichte der nächsten Schicht. 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 das Ergebnis der ersten Anwendung von Formel 8. Durch diesen Sachverhalt 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 berechnet.

Der in Formel 7 und 8 enthaltene Parameter zeigt auch auf wie schnell ein Netz lernt. Dies wird in **Abbildung ?** aufgezeigt. Dabei wurde, wie auch schon im gesamten Kapitel, die Sigmoid Funktion dargestellt, mit der benötigten ersten Ableitung der Funktion. Dabei ist zu sehen, dass der Anstieg in der Nähe von null und eins, gegen 0 geht. Der geringe Anstieg in diesen Bereichen sorgt dafür, dass das Neuron „*gesättigt*“ ist und nur noch langsam oder, wie bei der ReLU-Funktion durch die „*harte*“ 0, gar nicht mehr lernt.

**Ein Bild, das Himmel, draußen, Wand, Boot enthält.

Automatisch generierte Beschreibung**

Abbildung : Erste Ableitung der Sigmoid Aktivierungsfunktion

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

Formel : Formel zum Ändern der Gewichte

γ stellt die vom Benutzer gewählte Schrittweite beim Lernen da und ist der Gradient der Fehlerfunktion. Deshalb 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 YOLOv3, 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 gehören. (Redmon & Farhadi, 2018) Aus Gründen der fehlenden Rechenleistung des Erstellers dieser Arbeit wird auf andere und komplexeren Verfahren verzichtet.

# 5.4.1. You only look once (YOLO)

YOLO ist ein Algorithmus 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. YOLO arbeitet auf dem gesamten Bild. Zuerst wird das Bild in ein SxS-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 eine 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 Segmentierungsalgorithmus von den vorzustellenden Algorithmen. 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 vorgestellte, Algorithmus SSD und nur etwas schlechter wie der Algorithmus aus Kapitel 5.4.3.. Jedoch hat YOLOv2 ein Problem bei der Erkennung von kleineren Objekten in einem Bild. Bei YOLOv3 ist dieser Sachverhalt anders herum. YOLOv3 ist besser bei der Erkennung von kleinen Objekten und schlechter bei mittleren und großen Objekten. (Redmon & Farhadi, 2018)

**ToDo: YOLOv4 noch Ergänzen**

# 5.4.2. Single Shoot Detector (SSD)

SSD arbeitet wie YOLO indem es ein Gitter über das Bild legt, welches das Bild in gleichgroße Teile unterteilt. Für jeden Bereich des Bildes werden verschieden große und frei wählbare *Anchor Boxes* deklariert. Jede Box besitz dabei *K+1* Kategorien. *K* steht hierbei für die Anzahl der Klassifikationsgruppen und *+1* weil eine Anchor Box ohne Objekt mit 0, was für die Klassifikation als Hintergrund steht, Kategorisiert wird. (Zhang, C. Lipton, Li, & J. Smola, 2020) Die Anchor Boxes können jede beliebige rechteckige Form haben. 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.

Ein NN, welches mit SSD arbeitet, besitzt ebenfalls einen anderen Aufbau als ein Netz was mit YOLO arbeitet. SSD-Netze haben am Anfang ein NN, welches Semantische Informationen extrahiert. Dieses ist aufgebaut wie jedes andere CNN nur ohne dem voll vermaschten Teil. Der zweite Teil sind die SSD-Schichten. Die Schichten werden an den ersten Netzteil gehangen und die Ausgabe der SSD-Schichten werden, von dem folgenden voll vermaschten Netzteil, als Bounding Box und Klassifizierung des erkannten Objektes interpretiert. An den voll vermaschten Netzteil werden die Ergebnisse der zwei Teilbausteine geschickt. Ein wichtiger Hinweis ist auch, dass nicht nur Endergebnisse weitergegeben werden müssen. Es können auch Zwischenergebnisse der zwei beschriebenen Netzteile an den voll vermaschten Teil weitergegeben werden. (ArcGIS, 2019)

**Ein Bild, das Karte enthält.

Automatisch generierte Beschreibung**

Abbildung : 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

In obiger Abbildung ist erkennbar, das SSD in allen Fällen schneller ist als das im folgenden Kapitel vorgestellte Faster R-CNN. Jedoch ist die Durchschnittliche Genauigkeit der erzeugten Bounding Boxes 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 benutzt. Die erste Anwendung ist die Vorbestimmung von möglichen Objekten in einem *Region Proposal Network* (RPN) und die zweite ist das Region of Interest-Pooling, auch RoI-Pooling genannt. Dieses skaliert die Größe der Feature Map auf die geforderte Größe des folgenden Klassifizierungsschichte. (Grel, 2017)

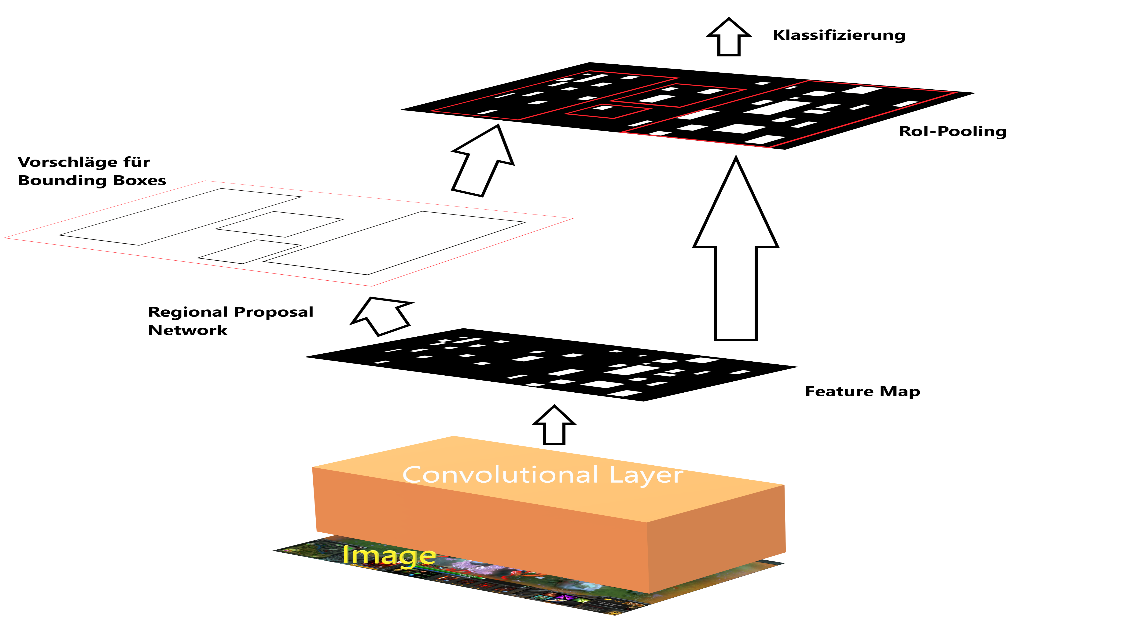
Das RPN bekommt die Feature Maps des CNN’s und gibt rechteckige Bereiche des Bildes, mit einer dazugehörigen Wahrscheinlichkeit zu welcher Gruppe dieser Bereich gehört, zurück. Um dies zu erreichen wird ein *nxn* große Convolutional Schicht erstellt, welches sich über die Feature Map schiebt. Das Ergebnis 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 1x1 Convolutional Schicht umgesetzt. (Grel, 2017) (Ren, He, Girshick, & Sun, 2016)

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

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

**Abbildung ?** zeigt den Aufbau eines Faster R-CNN um den eben beschriebenen Aufbau abermals verständlich Darzustellen.

Das Faster R-CNN ist der langsamste aber genaueste von den drei vorgestellten Algorithmen, was abermals in **Abbildung ?** zu sehen ist. (Huang, et al., 2017) Die geringere Geschwindigkeit und die höhere Genauigkeit der Bounding-Boxes entsteht durch die Arbeitsweise des Netzes. Die oben beschriebene Arbeitsweise zeigt auf, dass Faster R-CNN’s das Bild zweimal verarbeiten, um die endgültigen Bounding-Boxes zu bestimmen. YOLO und SSD gehen nur einmal über das Bild.

Dadurch wird Zeit und Rechenleistung gespart. Des Weiteren ist der Aufbau eines Faster R-CNN komplexer als die zwei anderen, was wiederum die Rechenzeit und den Speicherbedarf anhebt. (Huang, et al., 2017)

# 6. Realisierung der Pattern Erkennung mittels einem Convolutional Neuronal Network

Nach dem in den vorherigen Kapiteln Grundlagen der einzelnen Grundbausteine der Pattern Erkennung mit einem CNN erklärt wurden, geht es in diesem Abschnitt um die Realisierung der einzelnen Zwischenschritte. Die Umsetzung des Projektes wird in Python geschehen.

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 pro Lied 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 Drum und der Clap zu erkennen. Der Bass, Drum und die Clap sind selbst ohne CNN zu erkennen, wie **Abbildung ? a und c** beweist.

Ein Bild, das drinnen, Monitor, Bildschirm, Fernsehen enthält.

Automatisch generierte Beschreibung

Abbildung : a) Spektrum Highpass-Filter b) Spektrum Bandpass-Filter c) Spektrum Lowpass-Filter

Das Spektrum des Middle-Pass-Filters wird der schwierigste Teil in dieser Klassifikation. In diesem Frequenzbereich befinden sich die restlichen Instrumente. Somit überlagern sich in diesem Spektrum auch die Frequenzen der Instrumente, was das Spektrum in **Abbildung ?** b beweist. Dadurch wird es nicht mehr möglich sein die erkannten Pattern einem Instrument zuzuordnen. Somit soll das CNN lediglich wiederholende Muster im Spektrum erkennen. Das Erkennen von Mustern wird sich somit nicht auf einzelne Instrumente beschränken, wodurch im Bandpass-Spektrum lediglich sich wiederholende Blöcke angezeigt werden.

Zum Schluss werden die erkannten Pattern mithilfe von Bounding Boxes angezeigt.

# 6.1. Realisierung des Low-/Band-/High-Pass-Filter

Für die Umsetzung der drei Filter, wird das aus der Pythonbibliothek stammende Packet *aubio* verwendet. Dieses ist unter anderem zum Einlesen der *.wav* zuständig. Da es in diesem Packet keine Funktionen für einen Low-/Band-/High-Pass-Filter gibt, muss für dessen Umsetzung ein Biquad-Filter benutzt werden. 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 den Biquad bestimmt werden. Dafür werden mehrere Online Rechentools für die Berechnung der Biquad-Koeffizienten verwendet und die Ergebnisse 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).

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

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

Berechnet wurden folgende Koeffizienten:

* a0=1
* a1=-1,96929513
* a2= 0.96981555
* b0=1,30104102\*
* b1=2,60208205\*
* b2=1,30104102\*

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:

* =1
* =-0,77321399
* = 0.27545624
* =0,12556056
* =0,25112113
* =0,12556056
* =1
* =-1,95624013
* = 0.95715934
* =0,97834987
* =-1,95669974
* =0,97834987

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

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

Berechnet wurden folgende Koeffizienten:

* a0=1
* a1=-0,77321399
* a2= 0,27545624
* b0=0,51216756
* b1=-1,02433511
* b2=0,51216756

Nachdem die Koeffizienten bestimmt sind, sind folgende Filterfunktionen realisiert worden.

**BILD EINFÜGEN DER KURVENVERLÄUFE**

Da folgende Funktionen keine harten Kanten an Grenzen aufweisen, werden mehrere Biquads mit gleichen Koeffizienten hintereinander implementiert. Das führt dazu, dass die Grenzen der Filter härter werden. (Redmon N. , Biquads, 2003)

Nachdem die Filter auf das Musikstück angewendet wurden, ist ganz klar zu hören, dass die Filter funktioniert haben. Ob die Grenzen an den gewünschten Frequenzen liegen, lasst 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 die diese erst erstellt werden. Dafür wird das Programm *Sonic Visualizer* verwendet.

Sonic Visualizer ist ein kostenlosen Open Source Programm der Queen Mary Universität London, mit welchen Musik mittels einer Vielzahl von Tool analysiert werden kann. (Cannam, Landone, & Sandler, 2019)

Die erzeugten Spektren der MP3-Dateien sind vergleichbar zu **Abbildung ? a,b und c.** Bei der Kontrolle der exportierten Bilder der Spektren ist die Grenze der Filter klar erkennbar. Somit haben die in Kapitel 6.1. erstellten Filter wie geplant funktioniert. 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 einem NN beim Lernen zu helfen. Dabei werden die zu erkennenden Bilder mit rechteckigen Bounding Boxes umrandet. Die für diese Arbeit erstellten Bilder wurden mittels der Software *Yolo Label* bearbeitet. 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.

**BILD EINFÜGEN LABEL + TXT**

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

# 6.4. Erstellen des Convolutional Neuronal Networks

Bevor das Netz erstellt werden kann muss abgewägt werden, welches Netz 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. Die Wahl für *YOLOv4* wurde auf Grund der hohen Genauigkeit und den kurzen Berechnungszeiten gefällt wurden. Die Berechnungszeiten sind so gut, dass die Klassifizierung in Echtzeit in bis zu 120 FPS möglich ist. (Bochkovskiy, Wang, & Mark Liao, YOLOv4: Optimal Speed and Accuracy of Object Detection, 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 kein Bestandteil dieser Arbeit darstellen, 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 simpel. Es muss nur das Git-Repository geklont werden und mittels make erstellt werden. Im Anschluss müssen zwei Textdateien, in welchem die Test- und Evaluierungsdaten aufgelistet sind, und eine Config-, .data, und .names Datei angepasst werden.

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

Mit folgenden 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 Stunde bis zu einigen Tagen, je nachdem ob mit einem 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 aufgezeigt.

# 7. Pattern Erkennung mithilfe von Rhythmus und Melodie

Nachdem in Kapitel 6 die Spektren der einzelnen Musikstücke für die Erkennung von Pattern benutzt wurden, werden hier 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 Fünf-Dimensionale-Set der Noten eines Musikstückes. Bei der String basierte Pattern Erkennung und die Pattern Erkennung mithilfe von Matrizen werden die Musikstücke mittels MIDI-Dateien eingeladen. In den MIDI-Dateien sind unter anderen Informationen zu den gespielten Noten zu finden. Die Informationen zu den gespielten Noten beinhalten dabei 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 zeitliche Abhängigkeiten.

Alle in dieser Arbeit vorgestellten Algorithmen, welche mit der Geometrischen Pattern Erkennung arbeiten, verwenden ein Fünft-Dimensionales Punkt Set. Das Punkt Set wird im nächsten Schritt so verarbeitet, dass diese in ein zwei oder Dreidimensionales Koordinatensystem eingetragen werden können. (Meredith, 2006)

# 7.1. String basierte Patternsuche

Die String basierte Patternsuche wird ebenfalls 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 geringfügig 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 für zwei gleichzeitig gespielte Noten in zwei Varianten in der MIDI-Datei vorkommen und somit die Reihenfolge der Noten vertauscht wurde. Der Algorithmus berechnet zuerst die maximal Patternlänge. Diese Beträgt . Danach wird mit jeder Länge einmal das Musikstück rekursiv für jede Stelle durchgegangen. Die Rekursion besteht in der Änderung der Position um eine Stelle. Angefangen wird mit dem größtmöglichen Pattern bis zu einer Patternlänge von 3 Noten. Wenn der Algorithmus gestartet wird, wird die aktuelle Position und die aktuelle Patternlänge zur Bestimmung des momentanen zu untersuchenden Patterns verwendet. Im Anschluss wird die erste Stelle des Patterns mit der Position verglichen. Wenn die beiden Positionen übereinstimmen wird die zweite Position des Patterns mit der Position verglichen und das solange bis die beiden Stellen nichtmehr übereinstimmen. Wenn jedoch die erste Stelle nicht übereinstimmt, wird in den Fall des Keychange oder der vertikalen Spiegelung gegangen. In diesen beiden Fällen werden die aktuelle Position und die nächsten beiden Positionen miteinander verglichen.

Bei dem Keychange werden nur die Abstände zwischen der ersten/zweiten/dritten Position des Patterns und der aktuellen/nächster/übernächster 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 das bei der vertikalen Spiegelung eine gemeinsamen 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 der nächsten zwei Stellen 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 diesem sind enthalten welches Pattern gefunden wurde, welche Länge das gefundene Pattern aufweist und an welcher Stelle im Track dieses gefunden wurde. Mit diesen drei Eigenschaften kann der Anfang des Patterns ohne Probleme gefunden werden.

Wie gut die String basierte Patternsuche gegenüber den anderen Arten der Pattern Erkennung ist wird in Kapitel 9 aufgezeigt.

# 7.2. Pattern Erkennung mithilfe von Matrizen

Die Pattern Erkennung mit Matrizen, wurde wie auch schon die Pattern Erkennung 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 der momentan zu bearbeitendem Track geschrieben. Wenn dies geschehen ist entsteht eine Unterteilung an den Zellen . Die untere Hälfte der Matrix ist dabei nicht verwendbar, da dies einen 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 zu sehen.

**BILD EINFÜGEN DER MATRIX**

Wenn die Matrix initialisiert ist, kann der Algorithmus mit der Berechnung anfangen. Dabei wird durch jede Zelle der Matrix gegangen und analog zu der Pattern Erkennung mithilfe von Strings bearbeitet. Bei gleichen MIDI-Werten einer Zelle (x,y) erhält diese den Wert . Das gleiche geschieht bei einem Keychange und der vertikalen Spiegelung. 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. Der erste Startpunkt des Patterns besteht aus den Anfangskoordinaten einer diagonalen Kette und das Pattern besteht aus den x- oder y-Werten der gesamten Kette. **Abbildung ?** macht diesen Sachverhalt noch einmal deutlich.

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

<http://hss.ulb.uni-bonn.de/2013/3271/3271.pdf> (darunter alle von dem Link ab 6.2 interessant)

Stringbasierte Erkennung und geometrische Erkennung (Seite 101 Quellen anschauen) und beides zusammen (102 oben)

<https://www.researchgate.net/profile/Jia_Lien_Hsu/publication/221615538_Efficient_Repeating_Pattern_Finding_in_Music_Databases/links/54e614280cf2bff5a4f29302.pdf>

<http://pdfs.semanticscholar.org/f6ce/8e49f1987d927be91c99e82458c415266c89.pdf>

# 7.3. Vektorgeometrische Pattern suche

# 7.3.1. Fünf-Dimensionales Punkt Set

Das fünf-dimensionale Punkt Set besteht, wie der Name nahelegt, aus fünf Eigenschaften der Musik. Der erste Wert, stellt eine der wichtigsten Eigenschaften der Musik dar. Dieser Wert ist die Zeit. Wie die Zeit im ersten Wert dargestellt wird ist frei wählbar. Die Darstellung kann in Millisekunden oder auch in dem Zählen der Noten geschehen. Der zweite Wert ist die Höhe der gespielten Note. Die Höhe der Note befindet sich im Intervall (21, 128), welches die Range der MIDI-Noten darstellt. DRITTE ELEMENT DIATONIC PITCH

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

(Erst ab drittens Interessant!! Zweidimensional Aufbereiten <http://drops.dagstuhl.de/opus/volltexte/2006/652/pdf/06171.MeredithDavid.Paper.652.pdf> Plus Quellen, Chromatische Scala für Musik(Wert 2 der Fünf Dimensionales Punktset)

# 7.3.2. SIA

SIA steht für Structure Induction Algorithmus. Dieser Algorithmus nimmt das im vorherigen Kapitel beschriebene fünf-dimensionales Punkt Set als Vektor. 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 wichtig sein für genauere Untersuchungen von Musik.

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 **Abbildung ?** aufzeigt.

**ABBILDUNG EINFÜGEN KOORDINATENSYSTEM SIA**

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

**ABBILDUNG EINFÜGEN PATTERN+VEKTOREN**

Nachdem alle Vektoren berechnet sind, müssen diese lexikografisch sortiert werden. Die Sortierung kann dabei mit beliebigen Sortieralgorithmen durchgeführt werden. Wenn alle errechneten Vektoren sortiert wurden, sind alle Pattern ablesbar indem alle Werte eines lexikografischen Eintrages abgelesen werden. Jedoch können einige Werte in einem Eintrag nicht zu dem Pattern gehören, da diese Zufällig in dem Musikstück vorkommen können.

# 7.4. Umsetzung der Vektorgeometrischen Pattern suche

Die Vektorgeometrische Patternsuche wurde in Python umgesetzt. Zum das Einlesen der MIDI-Noten wurde kein Package verwendet, da in MIDI-Dateien keine Zeitangaben getroffen werden müssen. 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 Notennummer in die Datei eingetragen wird. Es ist dabei egal ob die einzelnen Tracks vermischt oder nacheinander eingetragen werden. Der gewählte Sortieralgorithmus ist *Quick-Sort*. Quick-Sort wurde gewählt, da die Implementierung trivial ist und er eine Laufzeit von im Average-Case aufweist. (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 zu der Genauigkeit sind in Kapitel 9 vorzufinden.

# 8. Pattern Erkennung mittels SIA und der anschließenden Auswertung durch ein CNN mit YOLOv4

Der letzte vorzustellende Weg zur Pattern Erkennung ist eine Kombination aus Vektorgeometrischer Erkennung und der Erkennung mittels eines CNN.

Die Kernidee hierbei besteht darin, dass die bildliche Darstellung von SIA Koordinatensysteme erzeugt, in welchen mit dem bloßen Auge und ohne großer musikalischer Ausbildung Pattern erkannt werden können. Da die Pattern automatisch erkannt werden sollen, werden die Bilder der erstellten Koordinatensystem an ein CNN weitergegeben. Dieses soll die Pattern dann 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. Als Eingabe wird abermals das Tripel aus Kapitel 7.4. per Numpy in ein Koordinatensystem eingetragen und gespeichert. Numpy erzeugt im folgenden Koordinatensysteme wie jenes in **Abbildung ?(in 7.3.)**. Anschließend wird ein CNN mit YOLOv4 darauf trainiert Pattern in diesen Dateien zu erkennen. Für das Training den 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 ersten erstellten CNN dieser Abschlussarbeit.

Wie gut oder schlecht diese Methode im Gegensatz zu den anderen Methoden funktioniert kann folgenden Kapiteln entnommen werden.

# 9. Vergleich der unterschiedlichen Verfahren der Pattern Erkennung

# 9.1. Anwendungsbereiche hinsichtlich Musikgenre

# 9.2. Rechenzeit

# 9.3. Anzahl der gefundenen Pattern

# 10. Fazit

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literaturverzeichnis

ArcGIS. (2019). *How single-shot detector (SSD) works?* Abgerufen am 31. Januar 2020 von 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. Abgerufen am 17. Januar 2020 von Neuronale Netze: http://www.neuronalesnetz.de

Bochkovskiy, A. (28. April 2020). *Github*. Von AlexeyAB/darknet: https://github.com/AlexeyAB/darknet abgerufen

Bochkovskiy, A., Wang, C.-Y., & Mark Liao, H.-Y. (23. April 2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. Von https://arxiv.org/pdf/2004.10934.pdf abgerufen

Brownlee, D. J. (9. Januar 2019). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Abgerufen am 17. Januar 2020 von Machine Learning Mastery: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Brownlee, D. J. (19. August 2019). *Crash Course in Convolutional Neural Networks for Machine Learning*. Abgerufen am 17. Januar 2020 von Machine Learning Mastery: https://machinelearningmastery.com/crash-course-convolutional-neural-networks/

Brownlee, D. J. (26. September 2019). *How Do Convolutional Layers Work in Deep Learning Neural Networks?* Abgerufen am 16. Januar 2020 von Machine Learning Mastery: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

Brownlee, D. J. (6. August 2019). *How to Avoid Overfitting in Deep Learning Neural Networks*. Abgerufen am 17. Januar 2020 von 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*. Abgerufen am 2. Januar 2020 von Sonic Visualiser: https://www.sonicvisualiser.org/doc/reference/1.3/en/

Cannam, C., Landone, C., & Sandler, M. (Dezember 2019). *Visualisation, analysis, and annotation of music audio recordings*. Abgerufen am 5. Februar 2020 von Sonic Visualizer: https://www.sonicvisualiser.org/

Drechsler, R. (5. März 2020). Klärung Musikalischer Grundlagen. (N. Mehlhose, Interviewer) Leipzig, Sachsen, Deutschland.

EasyCalculation.com. (kein Datum). BiQuadCalculator. Von https://www.easycalculation.com/physics/electromagnetism/biquad-calculator.php abgerufen

Glorot, X., Bordes, A., & Bengio, Y. (2011). *Deep Sparse Rectifier Neural Networks.* Abgerufen am 17. Januar 2020 von Proceedings of Machine Learning Research: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

Gonzales, R. R. (1996). *Theorie der neuronalen Netze.* Berlin: Springer. Abgerufen am 17. Januar 2020 von https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press. Von https://www.deeplearningbook.org/ abgerufen

Grel, T. (28. Februar 2017). *Region of interest pooling explained*. Abgerufen am 31. Januar 2020 von deepsense.ai: https://deepsense.ai/region-of-interest-pooling-explained/

Hermann, P. D. (2010). *Die Disktrete Fouriertransformation (DFT)*. Abgerufen am 19. Dezember 2019 von Technische Universität Wien: https://ti.tuwien.ac.at/cps/teaching/courses/dspv/files/DFT-FFT.pdf

Huang, J., Fathi, A., Rathod, V., Fischer, I., Sun, C., Wojna, Z., . . . Guadarrama, S. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. Abgerufen am 31. Januar 2020 von https://arxiv.org/pdf/1611.10012.pdf

Imagenet. (26. July 2017). *Large Scale Visual Recognition Challenge 2017 (ILSVRC2017)*. Abgerufen am 7. Januar 2020 von Imagenet: http://image-net.org/challenges/LSVRC/2017/results

Karpathy, D. A. (17. Oktober 2018). *CS231n Convolutional Neural Networks for Visual Recognition*. Abgerufen am 16. Januar 2020 von Github: http://cs231n.github.io/convolutional-networks/#add

Kriesel, V. (Februar 2013). Music Synchronization, Audio Matching, Pattern Detection, and User Interfaces for a Digital Music Library System. Bonn, Nordrein-Westfalen, Deutschland. Abgerufen am 18. März 2020 von http://hss.ulb.uni-bonn.de/2013/3271/3271.pdf

Krizhevsky, A., Sutskever, I., & Geoffrey, H. (2012). *ImageNet Classification with Deep Convolutional.*

Lutus, P. (26. Oktober 2019). BiQuadDesigner. Von https://arachnoid.com/BiQuadDesigner/index.html abgerufen

Math Works. (kein Datum). *Convolutional Neural Network: Drei Dinge, die Sie wissen sollten*. Abgerufen am 8. Januar 2020 von 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. Abgerufen am 15. März 2020 von https://drops.dagstuhl.de/opus/volltexte/2006/652/pdf/06171.MeredithDavid.Paper.652.pdf

MIDI Association. (2019). *MIDI Association*. Abgerufen am 26. Februar 2020 von 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. Abgerufen am 17. Januar 2020 von http://neuralnetworksanddeeplearning.com

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv. Abgerufen am 31. Januar 2020 von https://pjreddie.com/media/files/papers/YOLOv3.pdf

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (9. Mai 2016). You Only Look Once:. Von https://arxiv.org/pdf/1506.02640.pdf abgerufen

Redmon, N. (28. Februar 2003). *Biquads*. Abgerufen am 5. Februar 2020 von EARLevel Engineering: https://www.earlevel.com/main/2003/02/28/biquads/

Redmon, N. (13. Oktober 2013). Biquad calculator v2. Von https://www.earlevel.com/main/2013/10/13/biquad-calculator-v2/ abgerufen

Ren, S., He, K., Girshick, R., & Sun, J. (6. Januar 2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Abgerufen am 31. Januar 2020 von https://arxiv.org/pdf/1506.01497.pdf

Roberts, P. S. (2017). *University of Oxford.* Abgerufen am 31. Dezember 2019 von Lecture 7 - The Discrete Fourier: http://www.robots.ox.ac.uk/~sjrob/Teaching/SP/l7.pdf

Smith, S. W. (1997). *The Scientist & Engineer's Guide to Digital Signal Processing.* USA, Kalifornien: California Technical Pub.

Strick, H. K. (1. Juli 2012). *Joseph Fourier (1768–1830)*. Abgerufen am 18. Dezember 2019 von Spektrum: https://www.spektrum.de/wissen/joseph-fourier-1768-1830/1156113

Teufel. (9. Juli 2019). *Die Abtastrate – Tastend nach dem besten Sound*. Abgerufen am 2. Januar 2020 von Teufel: https://blog.teufel.de/abtastrate/#chapter2

Thormählen, P. D. (23. April 2018). *Multimediale Signalverarbeitung Frequenztransformation*. Abgerufen am 19. Dezember 2019 von Philipps Universität Marburg: https://www.mathematik.uni-marburg.de/~thormae/lectures/mmk/mmk\_3\_2\_ger\_web.html#1

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

Weisstein, E. W. (2. Februar 2015). *Apodization Function*. Abgerufen am 31. Dezember 2019 von Wolfram Math World: http://mathworld.wolfram.com/ApodizationFunction.html

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

Zhang, A., C. Lipton, Z., Li, M., & J. Smola, A. (2020). Dive into Deep Learning. Abgerufen am 31. Januar 2020 von https://d2l.ai

# 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