Inhaltsverzeichnis

[Abstract 4](#_Toc34047383)

[Danksagung 4](#_Toc34047384)

[1. Einleitung 4](#_Toc34047385)

[2. Arten von Pattern 4](#_Toc34047386)

[3. Die Fourier-Analyse 4](#_Toc34047387)

[3.1. Die diskrete Fouriertransformation 4](#_Toc34047388)

[4. Die Spektralanalyse 7](#_Toc34047389)

[5. Convolutional Neural Network 8](#_Toc34047390)

[5.1. Funktionsweise der einzelnen Bestandteile 9](#_Toc34047391)

[5.1.1. Convolutional Schicht 9](#_Toc34047392)

[5.1.2. Pooling Schicht 10](#_Toc34047393)

[5.1.3. Vollständig Vermaschtes Netzwerk 11](#_Toc34047394)

[5.2. Die ReLU-Aktivierungsfunktionen 11](#_Toc34047395)

[5.3. Backpropagation (vllt Seitenumbruch erzwingen) 13](#_Toc34047396)

[5.4. Segmentation Algorithmen 15](#_Toc34047397)

[5.4.1. You only look once (YOLO) 16](#_Toc34047398)

[5.4.2. Single Shoot Detector (SSD) 16](#_Toc34047399)

[5.4.3. Faster Region-Based Convolutional Neuronal Network (Faster R-CNN) 18](#_Toc34047400)

[6. Realisierung der Pattern Erkennung mittels einem Convolutional Neuronal Network 19](#_Toc34047401)

[6.1. Realisierung des Low-/Band-/High-Pass-Filter 20](#_Toc34047402)

[6.2. Erzeugen der Spektren 20](#_Toc34047403)

[6.3. Erstellen des Convolutional Neuronal Networks 21](#_Toc34047404)

[7. Pattern Erkennung mithilfe von Rhythmus und Melodie 21](#_Toc34047405)

[7.1. String basierte Pattern suche 21](#_Toc34047406)

[7.2. Pattern Erkennung mithilfe von Matrizen 21](#_Toc34047407)

[7.3. Vektorgeometrische Pattern suche 22](#_Toc34047408)

[7.3.1. Fünf-Dimensionales Punkt Set 22](#_Toc34047409)

[7.3.1.1. SIA 23](#_Toc34047410)

[7.3.1.2. SIATEC 23](#_Toc34047411)

[7.4. Umsetzung der Vektorgeometrischen Pattern suche 23](#_Toc34047412)

[8. Pattern Erkennung durch Zweidimensionale Darstellung von MIDI-Noten und der anschließenden Auswertung durch ein Convolutional Neuronal Network 23](#_Toc34047413)

[9. Vergleich der unterschiedlichen Verfahren der Pattern Erkennung 23](#_Toc34047414)

[9.1. Anwendungsbereiche hinsichtlich Musikgenre 23](#_Toc34047415)

[9.2. Rechenzeit 23](#_Toc34047416)

[9.3. Anzahl der gefundenen Pattern 23](#_Toc34047417)

[10. Fazit 23](#_Toc34047418)

[Abbildungsverzeichnis 23](#_Toc34047419)

[Tabellenverzeichnis 23](#_Toc34047420)

[Literaturverzeichnis 23](#_Toc34047421)

[Eidesstattliche Erklärung 26](#_Toc34047422)

# Abstract

# Danksagung

# 1. Einleitung

# 2. Arten von Pattern

# 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 : 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)

# 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 abgewägt wird zwischen Genauigkeit und Rechenzeit. 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)

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**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.

**TODO: ABBILDUNG NOCHMAL BEARBEITEN, DA ROTEN BEREICHE IN OBERSTER SCHICHT NICHT SICHTBAR SIND**

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 verschiedene Filter gelegt werden. Die Filter sind dafür wichtig um die Größe der Bilder, welche in das CNN gegeben werden, zu verringern. Die benötigten Filter sind ein Low-Pass-Filter, ein Band-Pass-Filter und ein High-Pass-Filter. Aus diesem Grund werden pro Lied drei Klassifizierungen durch das Netz nötig sein. Als nächstes werden die Patternstreams der Beats per minute (BPM) und des Rhythmus des Musikstücks extrahiert. **(Unter Umständen nicht nötig)** Danach werden die Spektrumbilder des Low- und High-Pass-Filter und die Patternstreams der BPM und des Rhythmus an ein neuronales Netz gegeben, um die Pattern des Basses, der Drum und der Clap zu erkennen. Der Bass, Drum und die Clap sind gut zu erkennen, wie **Abbildung ? a und c** aufzeigt.

**Bild mit den drei Spektren einfügen**

Das Spektrum des Middle-Pass-Filters wird der schwierigste Teil in dieser Klassifikation, da in diesem Frequenzbereich die restlichen Instrumente liegen. 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. Die Klassifizierung welches Instrument das angezeigte Pattern spielt muss somit noch vom Endbenutzter bestimmt 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. , 2003)

Als nächstes müssen die Koeffizienten für den Biquad bestimmt werden. Dafür wird Matlab verwendet. **Hier dann weiter am Freitag auch mit aufsagen wo die Filter enden sollen**

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

**BILD EINFÜGEN**

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

# 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 Spektren erstellt werden. Dafür wird das Programm *Sonic Visualizer* verwendet.

Der 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 exportierten Bilder der Spektren ist die Grenze der Filter klar erkennbar. Somit haben die in Kapitel 5.1. 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. Erstellen des Convolutional Neuronal Networks

Das CNN für die Pattern Erkennung wird in Python mittels Tensorflow und Keras erstellt. **Hier dann weiter nach der Erstellung des Netzwerks**

# 7. Pattern Erkennung mithilfe von Rhythmus und Melodie

Nachdem im ersten Teil der Arbeit die Spektren der einzelnen Musikstücke für die Erkennung von Pattern benutzt wurden, werden im zweiten Teil 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 Pattern suche

Die String basierte Pattern suche ist der einfachste Algorithmus aller in Kapitel 7 vorgestellter Algorithmen.

# 7.2. Pattern Erkennung mithilfe von Matrizen

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

Pattern können auch in Subpattern unterteilt sein

Pattern ranking function?

Ganze Musikstück muss nicht unbedingt ein Pattern sein es können auch „sinnlose“ Filler dabei sein

Arten von Pattern (Seite 98)

Methode: Melody extraction (nimmt an das Melodie in der höchsten Note enthalten ist)

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>

correlative Matrix Verfahren

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

Baumstruktur (suffix tree) beim auswerten (auch schon in anderen Werken benutzt)

# 7.3. Vektorgeometrische Pattern suche

# 7.3.1. Fünf-Dimensionales Punkt Set

(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)

Algorithmen dafür unter anderen: SIA (MTPs), SIATEC, COSIATEC)

# 7.3.1.1. SIA

# 7.3.1.2. SIATEC

# 7.4. Umsetzung der Vektorgeometrischen Pattern suche

# 8. Pattern Erkennung durch Zweidimensionale Darstellung von MIDI-Noten und der anschließenden Auswertung durch ein Convolutional Neuronal Network

Mit Bildoperation aus Computergrafik bearbeiten <- vllt nicht nötig da Einstellungen im Sonic Visualizer die obsolet machen

Motif Viewer mal anschauen

<https://arxiv.org/ftp/arxiv/papers/1707/1707.02051.pdf>

Laplace Operator könnte hilfreich sein

Faster RCNN+YOLO?

# 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

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/

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

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

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

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/

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