FAKULTÄT FÜR
INFORMATIK

Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Institut für Simulation und Graphik

Feature Selektion und deren Selektionsstrategien für die Objekterkennung in Bildern zur Vertonung von Umgebungssituationen

Bachelorarbeit

Author:

Hannes Stocker

Prüfer und Betreuer:

Priv.-Doz. Dr.-Ing.habil. Dirk Joachim Lehmann

Magdeburg, 17.12.2018

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeitarbeit selbstständig und ausschließlich unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder einer anderen Prüfungsbehörde vorgelegt oder noch anderweitig veröffentlicht.

Unterschrift	Datum

Kurzfassung/ Abstract

Diese Bachelorarbeit befasst sich mit dem Training und der Vertonung einer automatisierten Objekterkennung mit Hilfe Neuronaler Netze. Nachdem Grundlagen zu Bildverarbeitung und Deep Learning präsentiert wurden, werden aktuelle Algorithmen präsentiert die in Echtzeit Objekte erkennen, sie Klassifizieren und eine Bounding Box um sie ziehen. Es wird selbst ein Modell zum erkennen von Tassen mithilfe der Tensorflow Object Detection API erzeugt und über detektierte Bounding Boxen sonifiziert. Am Ende wird durch eine Versuchsreihe gezeigt, dass Probanden in der Lage sind mit diesem System zu interagieren.

This bachelor thesis deals with the training and sonification of an automated object recognition system supported by neural networks. After presenting the fundamentals of image processing and deep learning, current algorithms are presented that recognize objects in real time, classify them and draw a bounding box around them. We create a model for detecting cups using the Tensorflow Object Detection API and sonicate it via detected bounding boxes. In the end, a series of experiments shows that test persons are able to interact with this system.

Abkürzungsverzeichnis

CNN	Convolutional Neural Network
FPS	Frames pro Sekunde
GPU	Graphics Processing Unit
NN	Neuronales Netz
R-CNN	Region- Convolutional Neural Network
RoI	Region of Interest
RPN	Region Proposal Network
TF	Tensorflow
YOLO	You Only Look Once

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Das menschliche Gehör ist eines seiner hoch entwickelten Organe. Es ist in der Lage binnen kürzester Zeit Informationen aufzunehmen und zu verarbeiten. Im Gegensatz zur visuellen Wahrnehmung, können wir mit dem Aufmerksamkeitskanal des Hörens, nicht nur Reize aus einer beschränkten Perspektive wahrnehmen. Er Hilft uns sogar, Dinge wahrzunehmen, die sich außerhalb unseren Blickfelds befinden. Eine Kombination aus visueller und auditiver Wahrnehmung lässt uns viel mehr Reize wahrnehmen, als würden wir uns nur auf eine der beiden Wahrnehmungsarten beschränken. Sind wir nicht mehr in der Lage Dinge zu sehen oder visuell wahrzunehmen, liegt es am Gehör möglichst wichtige Reize zu erfassen. Umso besser wäre es, das Gehör wäre in der Lage mit Gegenständen zu agieren, als würde man sie vor seinen Augen sehen.

Diese Arbeit strebt an, dieser Eigenschaft einen Schritt näher zu kommen. Durch aktuelle Algorithmen der Echtzeit Objekterkennung, sowie Standards des Deep Learning, wird der Grundstein dafür gelegt, mit erkannten Objekten in der Umgebung über rein akustische Reize zu interagieren.

1.2 Ziele der Arbeit

Ziel dieser Arbeit ist es Ein System zu erschaffen, das in der Lage ist eine selbst gewählte Objektklasse zuverlässig zu erkennen. Dieses System soll dann die erkannte Objektklasse so auf Töne abbilden, dass man in der Lage wäre rein akustisch mit 2D- Bilddaten zu interagieren.

1.3 Gliederung der Arbeit

Als erstes werden Grundlagen der Bildverarbeitung diskutiert, die für das Erkennen von Objekten notwendig sind. Im Anschluss gibt es eine Einführung in Deep Learn-

ing und Neuronale Netze. Daraus resultierend werden aktuelle Objekterkennungsalgorithmen vorgestellt und teilweise miteinander verglichen. Danach geht es um die Realisierung eines Objekterkennungssystems für eine selbst gewählte Objektklasse und der Sonifikation davon. Am Ende wird in einer Versuchsreihe aus 3 Versuchen getestet, ob Probanden in der Lage sind, rein über akustische Reize mit dem System zu interagieren.

2 Grundlagen

2.1 Darstellung digitaler Bilder

Um Objekterkennungsalgorithmen auf Bilddaten anzuwenden, ist es notwendig zu wissen, wie Bilder für den Computer verständlich dargestellt werden.

Für die digitale Darstellung eines Bildes verwendet man gewöhnlicherweise eine Matrix. Anders als bei einem normalen Koordinatensystem, liegt der Ursprung $(u = 0, v = 0)$ dabei in der oberen linken Ecke. Jedes Element der Matrix steht für ein Pixel (Bildpunkt) im Bild. Das erste Pixel trägt die Koordinate $I(0, 0)$. Für ein Bild mit einer Auflösung von $M \times N$ Pixeln existieren somit maximal $M - 1$ Spalten und $N - 1$ Reihen.

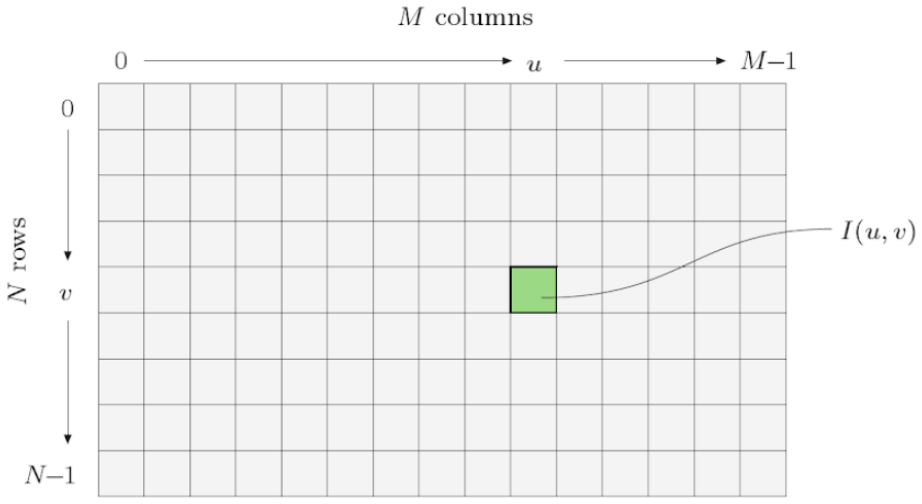


Abbildung 2.1: Digitales Bild als Matrix. Quelle: [1, S. 10]

Für jedes Pixel im Bild gibt es mindestens einen zugehörigen Wert. Beispielsweise besitzt beim typischen Graustufenbild jedes Pixel einen Intensitätswert. Dieser Intensitätswert steht für die Helligkeit und reicht von 0 (schwarz) bis 255 (weiß).

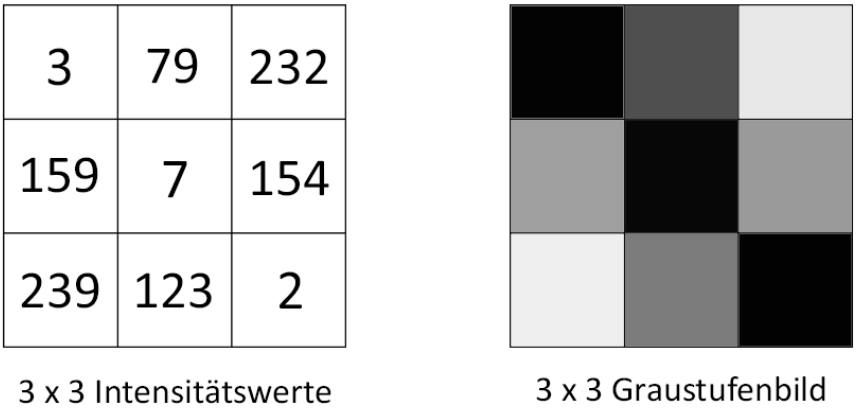


Abbildung 2.2: Gegenüberstellung von Intensitätswerten und der zugehörigen Graustufendarstellung eines 3×3 Bildes

Bei Farbbildern fasst ein Pixel mehrere Werte. Ein häufig verwendetes Beispiel dafür ist das RGB-Bild. Dort besitzt jedes Pixel für jede der drei Grundfarben rot, grün und blau einen eigenen Intensitätswert. Die Vereinigung dieser drei Werte ergibt am Ende die dargestellte Farbe.

Bei einigen Bildverarbeitungsalgorithmen macht es Sinn, das Farbbild vorher in ein Graustufenbild umzuwandeln und dadurch Berechnungszeit zu verkürzen. Da das menschliche Auge Farben in unterschiedlicher Stärke wahrnimmt, werden die Farben bei einer realitätsnahen Umrechnung auch unterschiedlich stark gewichtet.[1] Folgende Gleichung zeigt diese Umrechnung:

$$Grauwert = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{2.1}$$

Die in den Bildpunkten festgehaltenen Informationen bilden die Grundlage dafür, Merkmale von im Bild vorhandenen Objekten erkenntlich und für den Computer lernbar zu machen. Doch dafür bedarf es weiterer Methoden und Algorithmen die im Folgenden behandelt werden.

2.2 Faltung

Die sogenannte Faltung (Convolution) ist eine der grundlegenden Operationen der Bildverarbeitung. Durch sie ist es beispielsweise möglich, Kanten zu erkennen oder die Bild-

schärfe zu manipulieren. Faltung spielt außerdem eine wichtige Rolle bei moderner, automatisierter Objekterkennung.

Faltet man ein Bild, so bedeutet dies, man manipuliert jedes Pixel in Abhängigkeit seiner direkten Nachbarn.[2] Dies geschieht mit Hilfe einer Filtermaske (Filterkern). Die Filtermaske ist ebenfalls eine Matrix, bei der die verschiedenen Werte durch Gewichtungen dargestellt werden. Um die Faltung für einen Pixel im Eingangsbild durchzuführen, platziert man das Maskenzentrum über diesem Pixel und berechnet mit Hilfe der Gewichtungen den Pixelwert neu. Wie in Abbildung 2.3 zu sehen, ersetzt der neu berechnete Pixel den Eingangspixel im Ergebnisbild. Um das gesamte Eingangsbild zu falten, lässt man die Maske über jeden Pixel im Ausgangsbild iterieren.

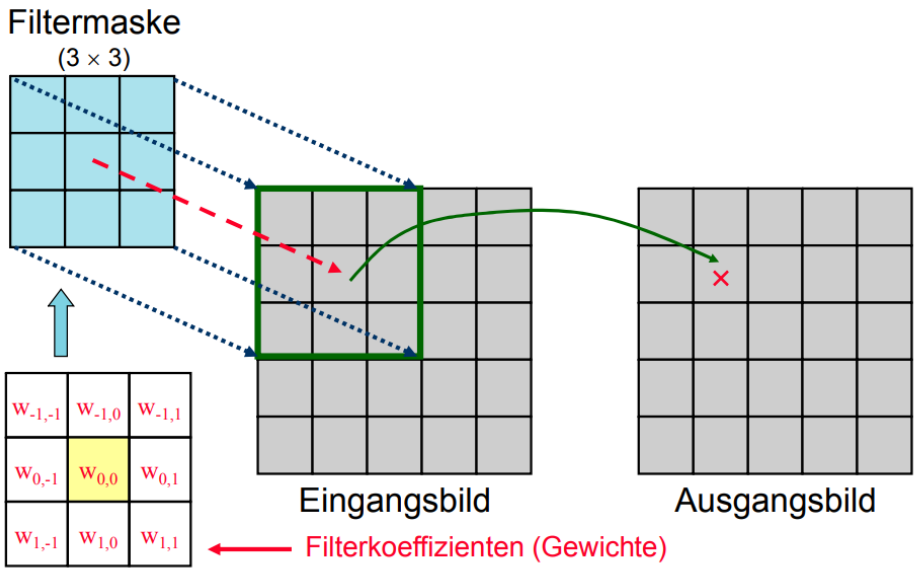


Abbildung 2.3: Vereinfachte Darstellung von Convolution. Quelle: [2]

Die Berechnung der diskreten Faltung lässt sich durch Formel 2.2 darstellen. In dem Fall bezeichnen wir $I_{i,j}$ als Eingangsbild, $O_{i,j}$ als Ausgangsbild, W als Maskengröße (in unserem Fall 3) und $M_{p,q}$ als Filtermaske selbst.

$$O_{i,j} = \sum_{p=(-\frac{W}{2})}^{\frac{W}{2}} \sum_{q=(-\frac{W}{2})}^{\frac{W}{2}} I_{i-p,j-q} \cdot M_{p,q} \tag{2.2}$$

Filtermasken besitzen immer eine ungerade Größe. Berechnen wir damit die äußeren Pixel eines Bildes, so liegt die Filtermaske teilweise außerhalb des Bildes und kann nicht auf gleiche Weise berechnet werden.

Durch verschiedene Lösungsansätze der Randbetrachtung, lässt sich dieses Problem umgehen. Die einfachste Möglichkeit ist, am Rand keinen Filter zu verwenden, sondern das

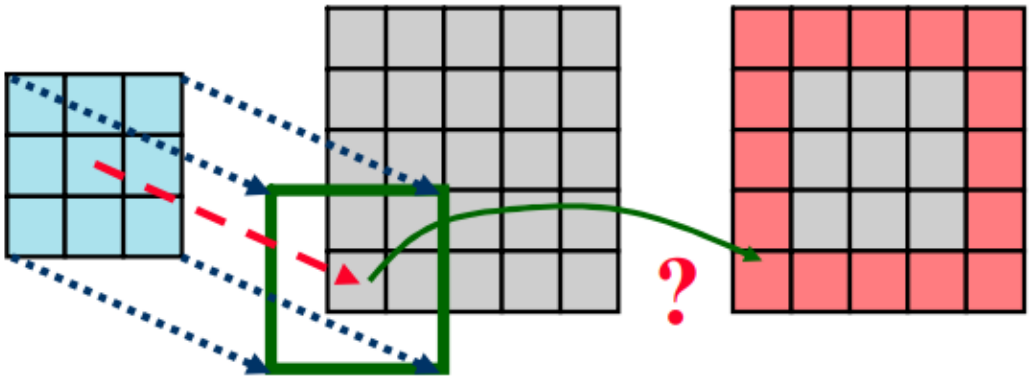


Abbildung 2.4: Convolution und Randbetrachtung. Quelle: [2]

Bild erst ab dem ersten Pixel zu falten, auf das sich die Maske anwenden lässt. Andere Methoden setzen das Bild künstlich am Rand fort. Last Value ist eine dieser Methoden. Dabei wird der originale Rand mit einem künstlichen Rand erweitert, der die gleichen Werte beinhaltet wie die Randpixel des Eingangsbildes. Der Filtermaske ist es so möglich auch die äußeren Pixel des Eingangsbildes neu zu berechnen. [2] Bei der Objekterkennung in Bildern haben die Randbereiche meist keine relevanten Informationen. Dort kann also auf Randbetrachtung verzichtet werden.

2.3 Deep Learning

Deep Learning ist ein Forschungsbereich von Machine Learning (Maschinelles Lernen). Hier wird versucht, die Fähigkeit des Menschens Dinge zu Lernen, abzurufen und auf Problemstellungen anzuwenden, annähernd auf Maschinen zu übertragen. Das Ziel von Deep Learning ist es, sich immer weiter der künstlichen Intelligenz zu nähern. Dabei geht es vor allem darum, mehrere Stufen und Abstraktionen zu erlernen, um Bild-, Ton- und Textdaten zu verstehen.[3] Der Computer Arbeitet mit einer Art Hierarchie einfacher Lösungsansätze, die es im Zusammenwirken möglich macht, einen komplexeren Lösungsansatz zu bilden.[4]
Im Falle dieser Arbeit bildet Deep Learning die Grundvoraussetzung für das Lernen visueller Objekteigenschaften, um Objekte wiederzuerkennen und ihre Position im Bild zu bestimmen.

2.3.1 Neural Networks

Das Menschliche Gehirn besitzt unzählige Neuronen mit mehreren Milliarden Verbindungen zwischen ihnen. Pro Hemisphäre im Gehirn hat der Mensch einen primären visuellen Kortex, welcher allein schon 140 Millionen Neuronen enthält. Der primäre visuelle Kortex ist jedoch nur einer von fünf visuellen Kortexen, die zunehmend immer komplexere Bildverarbeitung durchführen.[5]

Ein grundlegendes Beispiel für das Erfassen von Informationen aus Bilddaten ist das Erkennen von handgeschriebenen Ziffern. Innerhalb kürzester Zeit gelingt es dem Gehirn diese Ziffern zu erkennen und es fällt ihm leicht unterschiedliche Schreibweisen oder visuelle Abweichungen wahrzunehmen.

Abbildung 2.5: Von Hand geschriebene Ziffern, Zahlenfolge: 504192 Quelle: [5]

Möchte man einen Algorithmus schreiben, der in der Lage ist nicht nur eine vorgegebene Zahlenfolge dieser Art zu erkennen, sondern alle, so muss der Computer vorher in der Lage sein, jede Art von Handgeschriebener Ziffer richtig zu erkennen. Die Lösung für dieses Problem ist ein künstlich erzeugtes neuronales Netz.

Das einfachste Modell eines Neuronen ist das Perzeptron. Das Perzeptron bekommt beliebig viele binäre Eingangswerten x_1 bis x_n als Input. Jeder dieser Eingangswerte wird mit einem Gewicht, einer reellen Zahl w_1 bis w_n versehen. Mit Hilfe der Eingangswerte und Gewichte lässt sich ein Ausgangswert (output) berechnen.

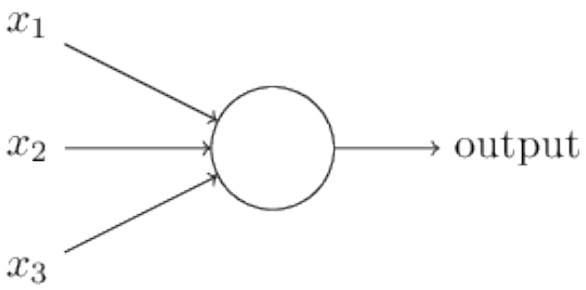


Abbildung 2.6: Perzeptron Beispiel mit drei Eingangswerten. Quelle: [5]

Die Gewichte stellen dabei Zahlen dar, die die Wichtigkeit der jeweiligen Eingabe für die Ausgabe ausdrückt. Der Ausgangswert des Perzeptrons ist entweder 0 oder 1. Welcher dieser beiden Ausgangswerte weitergegeben wird hängt davon ab, ob die Summe der Eingangswerte größer oder kleiner als der Schwellenwert des Perzeptrons ist. Dieser Sachverhalt lässt sich durch Ausdrucksweise 2.3 verdeutlichen.[5] Der Schwellenwert ist die Hauptinformation die das Neuron besitzt. Er ist ebenso, wie die Eingangswerte, eine reelle Zahl.

$$output = \begin{cases} 0 & wenn \sum_j w_j x_j \leq Schwellenwert \\ 1 & wenn \sum_j w_j x_j > Schwellenwert \end{cases} \tag{2.3}$$

Das Grundmodell des Neuronalen Netzes ist ein Layer Modell. An erster Stelle steht der Input Layer. Dieser beinhaltet alle Eingangsinformationen. Bei bildverarbeitenden Netzwerken kann der Input Layer zum Beispiel aus der Auflistung der im Bild vorhandenen Pixel und der dazugehörigen Intensitätswerte bestehen. Nach dem Input Layer kommen ein oder mehrere Hidden Layer. Die Hidden Layer setzen sich aus einer Anordnung von Neuronen zusammen. Die Anzahl der Neuronen kann variieren. Der letzte Hidden Layer führt zum Output Layer, welcher, wie in Abbildung 2.7 zu sehen ist, am Ende das Ergebnis wiedergibt.

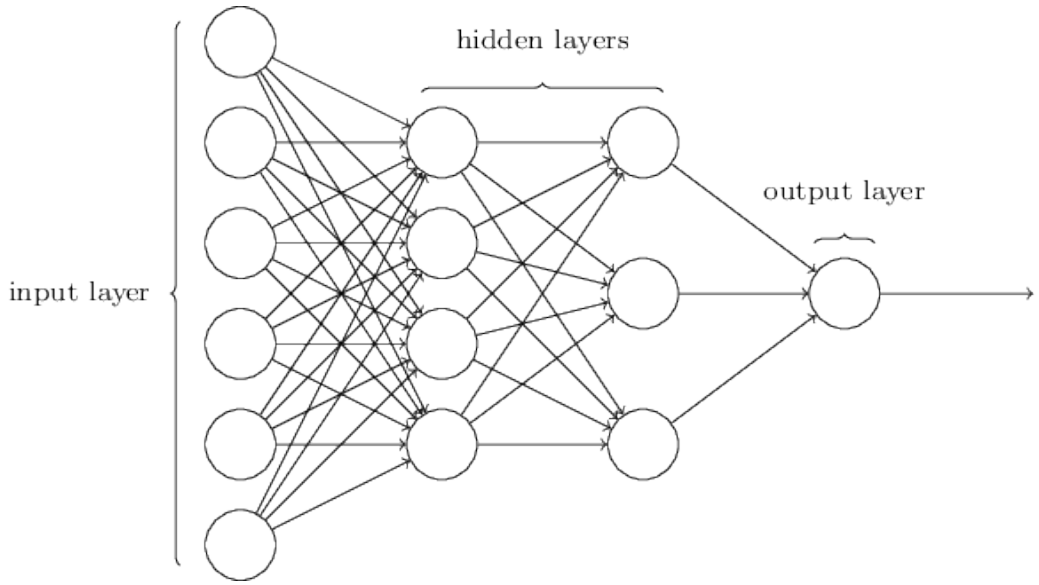


Abbildung 2.7: Neuronales Netz Beispiel mit zwei Hidden Layern. Quelle: [5]

Bei dem Beispiel der handschriftlichen Zahlen, würde der letzte Hidden Layer aus allen zehn Ziffern bestehen und das Neuron, welches die angenommene Zahl enthält, würde

seine Information an den Output Layer wiedergeben.

Die Richtung in der die Neuronen auslösen ist nicht immer gleich. Man unterscheidet in feed-foreward-Netze und rekurrente Netze [6]. Rekurrente Netze haben nicht nur vorwärts gerichtete Kanten, sondern besitzen ebenso Kanten, die in die Rückrichtung gehen. Somit kann die Ausgabe eines Layers, ein vorheriges Neuron oder eines der gleichen Schicht als Eingabe haben. Doch nicht nur die Anzahl der Layer oder die Auslöserichtung sind entscheidend für die Klassifizierung eines Neuronalen Netzes.

Es gibt verschiedene Arten von Lernmethoden. Die Methode die wir für das Training des neuronalen Netzes nutzen nennt sich Supervised Learning (kontrolliertes Lernen). Dabei handelt es sich um eine Methode, wo die korrekte Zuordnung der Trainingsdaten von vornherein bekannt ist.[5] Möchte man Beispielsweise ein Objekt im Bild erkennen, so gibt es für jedes Bild im Datensatz eine Label Datei, in der die korrekten Positionsdaten des Objekts bereits vorhanden sind. Angewandt wird das Supervised Learning bei mehrschichtigen feed-foreward Netzen. Das lernen über Fehler in einem Netz dieser Art wird als Error Backpropagation bezeichnet. Der Algorithmus berechnet die Fehlerdifferenz der Ausgabe des Netzes zur bereits bekannten richtigen Ausgabe. Dabei passt er die synaptischen Gewichte der Neuronen an, die proportional zum Produkt des Fehlersignals und der Eingabeinstanz des synaptischen Gewichts ist. Demnach lässt sich Error Backpropagation durch 2 Schritte beschreiben.[6]

Foreward Pass: In diesem Schritt bleiben die synaptischen Gewichte noch unverändert. Der Eingabevector (zum Beispiel ein Bildsignal) geht in das Netz hinein und durchläuft alle verbundenen Neuronen und Layer bis es am Ende als Outputsignal herauskommt. Die Ausgabe wird mit der gewünschten Antwort verglichen. Dadurch kann der Fehler (die Abweichung) vom Sollzustand ermittelt werden.

Backward Pass: Der Fehler wird nun auf vorhergehende Layer zurückgeführt. Nun berechnet man über den Fehler den lokalen Gradienten für jedes vorhandene Neuron und passt die Gewichte an.

Diese beiden Schritte werden so lang wiederholt bis das Neuronale Netz die gewünschten Ergebnisse liefert und der Fehler weitestgehend minimiert ist.

2.3.2 Convolutional Neural Networks

Moderne Objekterkennung arbeitet mit verschiedenen Formen von Convolutional Neural Networks (Konvolutionale Neuronale Netze, CNNs). Hier werden die bereits erklärten

Prinzipien der Faltung mit denen der Neuronalen Netze vereint. Der Hauptvorteil von CNNs ist es, dass das gesamte System zu Ende trainiert wird, von den einzelnen Pixeln bis zu den entgültigen Kategorien, wodurch die Notwendigkeit eines manuellen Entwurfs verringert wird. Der Nachteil ist, sie benötigen große Mengen an mit Labeln versehenen Trainingsdaten.[7]

Ein CNN besitzt im Bereich der Hidden Layer ein oder mehrere Convolutional Layer. Jedes Neuron dieses Layers repräsentiert einen Filter. Die Filter unterscheiden sich in den Werten der einzelnen Felder ihrer Matrix. So kann der Layer beispielsweise aus verschiedenenen Layern zur Kantenermittlung bestehen. Dabei ist als Input, wie in Abbildung 2.8, kein einzelner Pixel, sondern ein komplettes Bild pro Neuron gegeben.

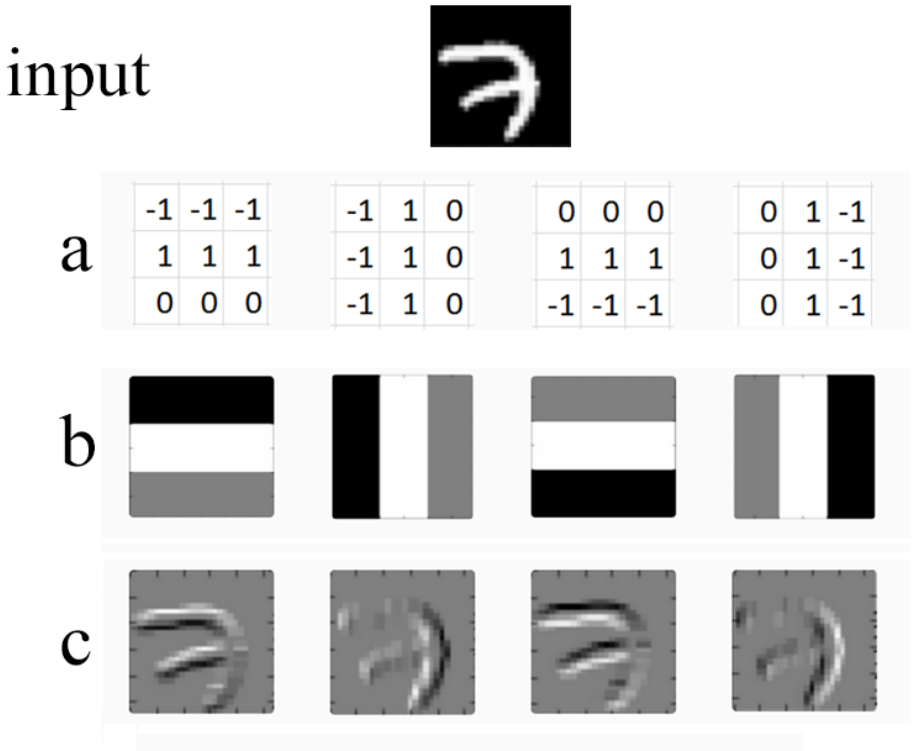


Abbildung 2.8: Input Bild: handschriftliche 7, a: Convolutional Layer aus 4 Filtern mit Zahlenwerten, b: Layer a mit zugehörigen Helligkeitswerten, c: output nach Anwendung der Filter auf das input Bild. Quelle: [8]

Je mehr Convolutional Layer wir durchlaufen, umso komplexer werden die erkannten Merkmale. Hinter jedem Convolutional Layer kommt üblicherweise ein Pooling Layer. Die Aufgabe dieses Layers ist es, die Auflösung der Eingaben zu reduzieren um die Anzahl der Gewichte zu mindern und somit den Rechenaufwand zu reduzieren. [9] Dies erfolgt meistens durch eine 2x2 Matrix, die jede 2x2 Teilmatrix des Eingangsbildes durch ihren Maximalwert ersetzt. Diese beiden Layer- Typen lassen sich Zusammenfassen zum sogenannten Feature Extractor. Die CNN- Features des letzten Pooling Layers werden

als feature-Vektoren an den classifier weiter gegeben. Dieser besteht aus einem typischen feedforeward-Neuronalen Netz aus Perzeptronen. dieses klassifiziert die übergebenen Feature und leitet sie schließlich zum Output weiter. Dieser Sachverhalt wird in Abbildung 2.9 illustriert.

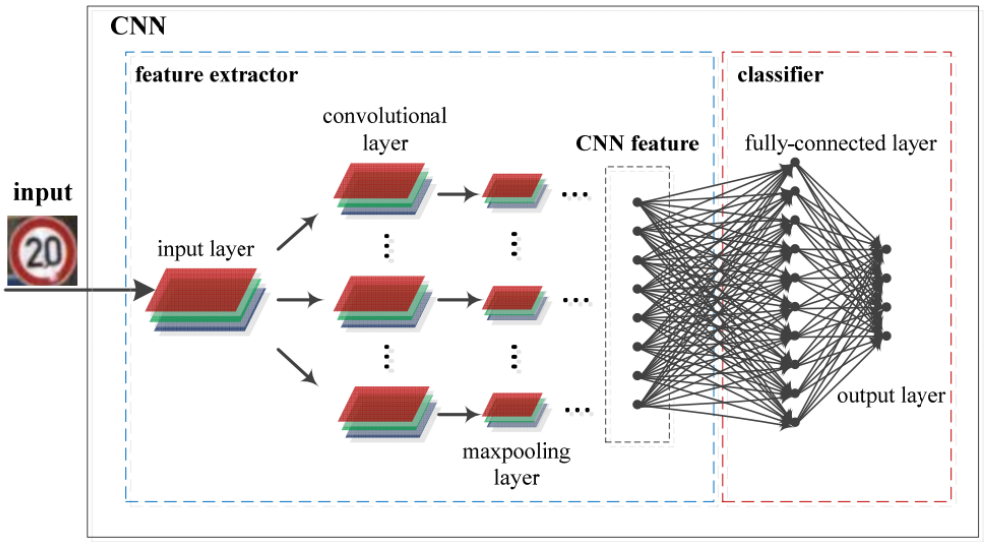


Abbildung 2.9: Aufbau eines CNNs am Beispiel eines Verkehrszeichen Bildes als Input Quelle: [9]

3 Stand der Technik

3.1 Effiziente Objekterkennungsalgorithmen

Im vorherigen Kapitel ging es unter anderem um Objektvorhersage. Um mit Positionsdaten von Objekten zu arbeiten, reicht es jedoch nicht aus zu wissen, dass sich das gesuchte Objekt im Bild befindet. Wir benötigen eine Positionsvorhersage. Folgende Algorithmen bieten genau diese Lösung. Mit ihnen lässt sich ein Modell Trainieren, welches Bounding Boxes (begrenzende Boxen) um gesuchte Objekte zieht und diese visualisiert. Sie unterscheiden sich in ihrer Genauigkeit, so wie in der Geschwindigkeit der Berechnungen. Dieser Abschnitt wird diese Algorithmen grundlegend vorstellen.

3.1.1 Faster R-CNN

Die Ausgangsform des Faster R-CNN Algorithmus ist der R-CNN Algorithmus. Bei R-CNN werden mit Hilfe von Selektiver Suche 2000 Region Proposals (Vorschläge für Regionen/Kandidaten) ermittelt. Dadurch wird der Input der zu berechnenden Bildbereiche limitiert. Die Selektive Suche generiert dabei Regionen möglicher Kandidaten und fasst ähnliche Regionen zu größeren zusammen. [10] Diese Kandidatenvorschläge werden dann gemeinsam in ein CNN eingespeist. Das CNN extrahiert die Merkmale und gibt diese als Output Layer weiter. Diese werden dann wiederum in eine SVM (Support Vector Maschine) eingespeist.[11] Dort wird der Kandidat auf das vorhanden sein des Objekts überprüft. Zusätzlich zu dieser Vorhersage werden vier Offset Werte (Versatzwerte) erzeugt, die die Genauigkeit der Bounding Box erhöhen.[12] So kann es beispielsweise sein, dass in einem Bildausschnitt ein Hund erkannt wird, aber sein Körper zur Hälfte fehlt. Die Offset Werte enthalten dann Informationen, die zur Verschiebung der Bounding Box benötigt werden könnten, sodass der Hund komplett innerhalb der Box ist.

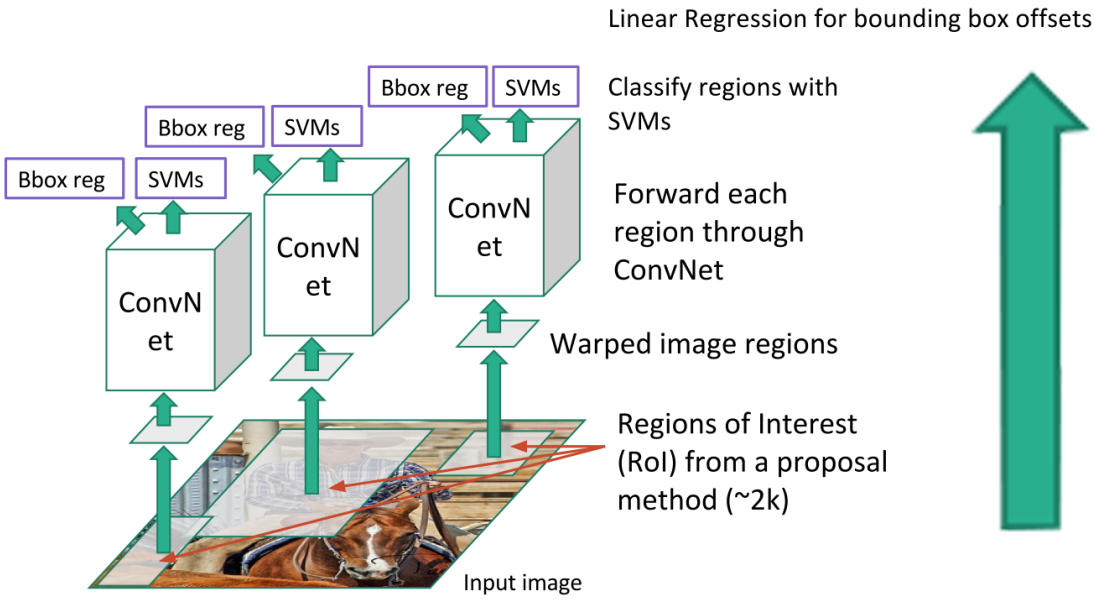


Abbildung 3.1: Vereinfachte Darstellung eines R-CNNs. Quelle: [12]

R-CNN ist noch keine Echtzeitvariante, da für jedes Bild rund 50 Sekunden Berechnungszeit benötigt wird. Außerdem ist die Trainingsdauer selbst bei der Einschränkung auf 2000 zu betrachtenden Regionen noch sehr hoch. [12]

Die erste Verbesserung zum R-CNN ist das Fast R-CNN. Dabei werden nun nicht mehr die möglichen Objektkandidaten an das CNN weiter gegeben, sondern das komplette Bild. Aus diesem Bild wird dann eine Convolutional Feature Map(konvolutionale Merkmalskarte) extrahiert.[11] Diese ist dafür zuständig Merkmale und zu identifizieren, beispielsweise das Erkennen eines kreisförmigen Objekts (siehe Abbildung3.2), und ihre Position zu bestimmen.[13]

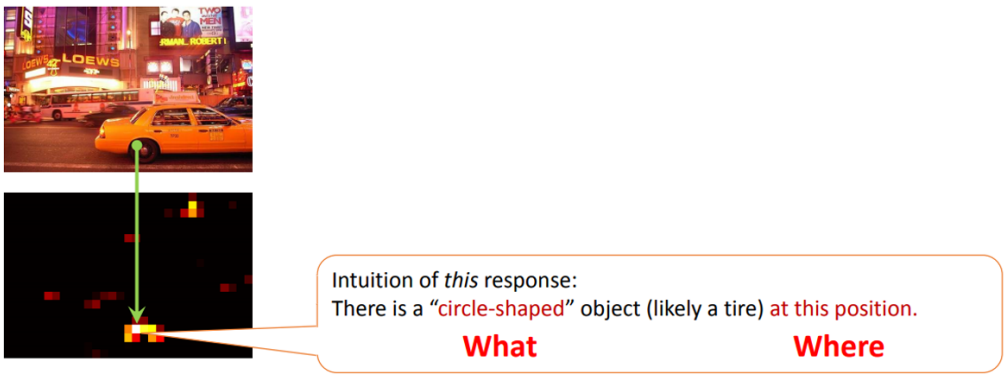


Abbildung 3.2: Feature Maps, Merkmale und ihre Lokalisierung. Quelle: [13]

Aus der Feature Map lassen sich die Regionen der möglichen Kandidaten identifizieren. Diese werden dann in zugehörige Quadrate gepackt. Mithilfe eines RoI-Pooling-Layers (Region of Interest-Pooling-Schicht) werden diese Quadrate auf eine festgelegte Größe gebracht, sodass sie als ein Vektor in einen Fully Connected Layer eingespeist werden können. Von da ausgehend wird durch einen Softmax- Layer die Klasse des Objekts, sowie die Versatzdaten für die Bounding Box vorhergesagt.[12]

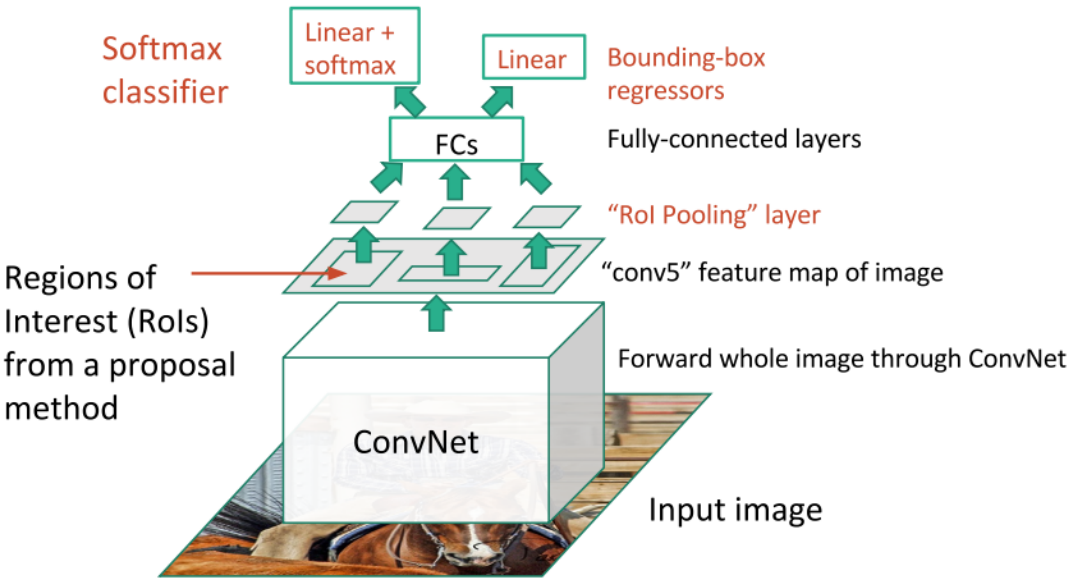


Abbildung 3.3: Vereinfachte Darstellung eines Fast R-CNNs. Quelle: [12]

Da wir beim Fast R-CNN nicht jedes mal 2000 Regionsvorhersagen durch das CNN schicken müssen, sondern die Faltungsoperation nur ein mal pro Bild durchgeführt wird, ist Fast R-CNN bedeutend schneller als RCNN. Die Trainingsdauer ist ca. fünf mal so schnell, die Testdauer sogar mehr als 20 mal schneller.[12] Bei mehr als zwei Sekunden Pro Bild können wir allerdings noch nicht von Echtzeiterkennung sprechen. Eine weitere Verbesserung wird benötigt.

So haben Shaoqing Ren u.a. 2016 Faster R-CNN veröffentlicht. Faster- RCNN funktioniert komplett ohne Selektive Suche. Um die Regionsvorhersagen zu bestimmen wird ein eigenständiges Netzwerk, das RPN (Region Proposal Network) verwendet. Die Regionen werden dann wieder durch ein RoI - pooling layer in ihrer Größe angepasst, um dem Bild in den Regionsvorhersagen Ojektklasse und Offset-Größen für Bounding Boxen zuzuordnen.[11]

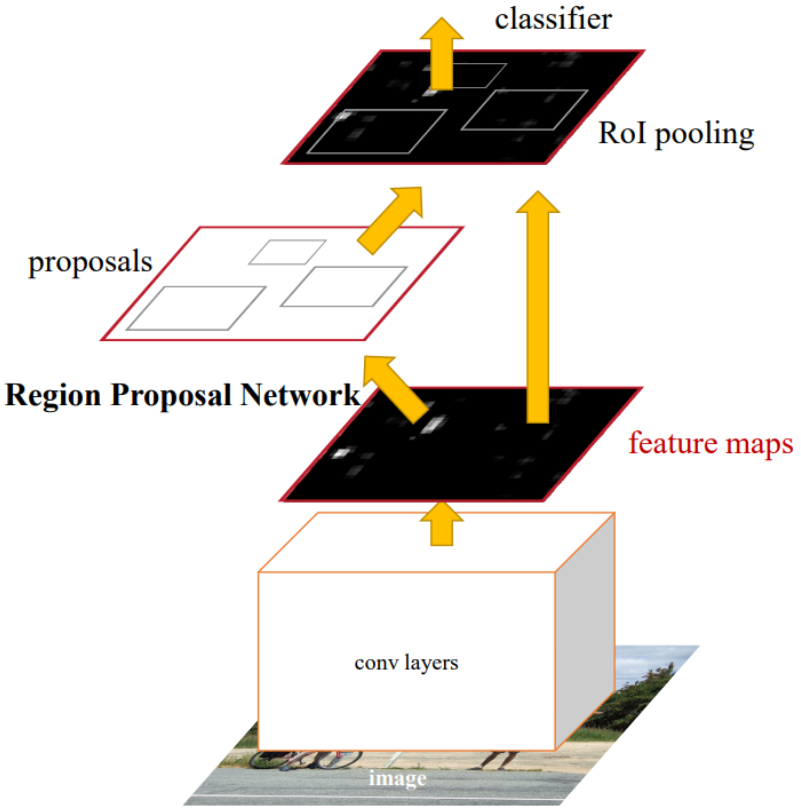


Abbildung 3.4: Vereinfachte Darstellung eines Faster R-CNNs. Quelle: [11]

Beim RPN wird ein kleines Netzwerk über den letzten ausgegebenen Convolutional Layer bewegt. Dieses Netzwerk nutzt als Input ein $n \times n$ Fenster der Input Feature Map. Dieses Fenster wird gleichzeitig in zwei fuly connected Layer eingespeist. Der erste Layer ist der box-regression Layer (reg), der zweite der box-classification Layer (cls). An jedem Fenster haben wir gleichzeitig mehrere Regionsvorschläge, welche durch ein Maximum k begrenzt werden. Somit besitzt der reg- Layer eine $4k$ - Kodierung für alle Koordinaten der k Bounding Boxen und die Ausgabe des cls- Layers $2k$ Scores, die die Wahrscheinlichkeit dafür angeben, ob sich ein Objekt in der Regionsvorhersage befindet oder nicht. Hier führt man den Begriff Anchor (Anker) ein. Ein Anchor ist als Referenzfeld zu sehen. Für k Vorhersagen gibt es auch k Referenzfelder. Der Anchor hat seinen Mittelpunkt im sich bewegenden Fenster. Er ist variabel in Seitenverhältnissen und Größe (Anzahl der Pixel). Die Anzahl k setzt sich dann aus Anzahl der Seitenverhältnisse multipliziert mit der Anzahl der Gößenvariationen zusammen. Für gewöhnlich werden hier drei Größen und drei Seitenverhältnisse gewählt, also $k = 9$.[11]

RPN und Fast-RCNN werden unabhängig voneinander trainiert. Die einfachste Lösung dafür ist das Alternating Training (Abwechselndes Training). Dabei wird zuerst das RPN trainiert, um die daraus resultierenden Vorhersagen zum Training des Fast R-CNNs

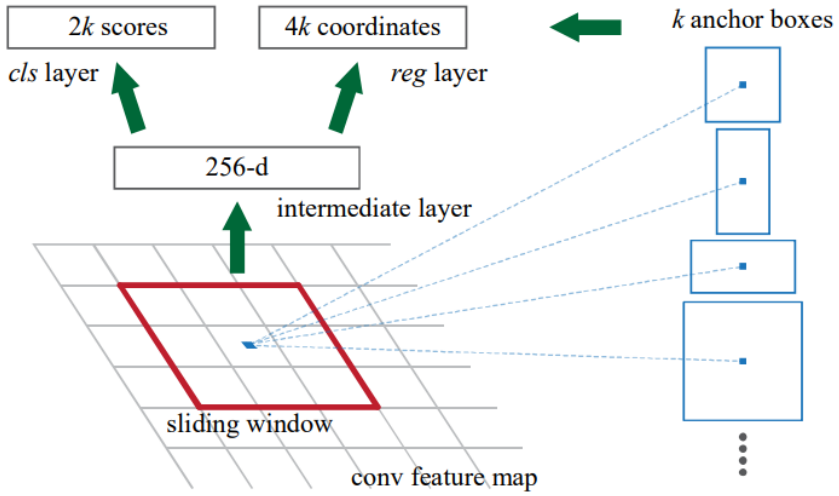


Abbildung 3.5: Vereinfachte Darstellung von RPN. Quelle: [11]

zu nutzen. Das von Fast- RCNN Resultierende Netzwerk wird dann wieder genutzt, um RPN zu initialisieren. Dieser Vorgang wiederholt sich bis der Trainingsvorgang das gewünschte Ergebnis erzielt oder die festgelegte Anzahl an Trainingsschritten erreicht wurde.[11]

Faster R-CNN ist mehr als 10 mal schneller als R-CNN. Ein Testdurchlauf dauert durchschnittlich nur noch 0.2 Sekunden. Damit lässt sich nun also die gewünschte Echtzeitobjekterkennung realisieren[1].[12]

3.1.2 YOLO

YOLO oder You Only Look Once ("du schaust nur ein mal hin") ist ein Objekterkennungsalgorithmus, der sich von den bisher betrachteten Algorithmen in der grundlegenden Herangehensweise unterscheidet. Anstelle von separaten Bildregionen betrachtet YOLO das komplette Bild und nutzt ein einziges NN um Bounding Boxen und Klassenwahrscheinlichkeiten vorherzusagen. Außerdem werden die Bounding Boxen aller Objektklassen gleichzeitig bestimmt.

Zu Beginn wird das Bild in ein $S \times S$ Gitter aufgeteilt. Für jedes Feld des Gitters werden B Bounding Boxen erzeugt. Das NN erzeugt für jedes Gitterfeld Werte der Klassenvorhersagen (confidence) und Offset- Werte für die Bounding Boxen. Fällt der Mittelpunkt eines gefundenen Objekts in ein Gitterfeld, so wird dieses Feld als mögliches Feld zur Erkennung des Objekts markiert. Wenn kein Objekt im Gitterfeld erkannt wurde sind die confidence-

[1] Die hier verwendeten Zeitangaben sind von 2016/ 2017. Durch stetig steigende Rechenleistung werden diese immer geringer. Sie dienen hauptsächlich als Vergleichswert.

Werte null. Während der Testzeit wird die bedingte Klassenwahrscheinlichkeit mit der confidence- Vorhersage der einzelnen Boxen multipliziert. Dadurch erhalten wir die spezifischen confidence- scores für jede der Bounding Boxen. Dadurch wird berechnet wie wahrscheinlich es ist, dass die gesuchte Klasse innerhalb der Box liegt und wie gut die vorhergesagte Box das Objekt in Größe und Position wiedergibt.[14] Die finale Ausgabe wird dann durch einen Schwellenwert bestimmt. Die Bounding Boxen, deren Klassenwahrscheinlichkeit über dem festgelegten Schwellenwert liegen (meist 0.7) werden, wie in Abbildung 3.6 zu sehen, angezeigt.

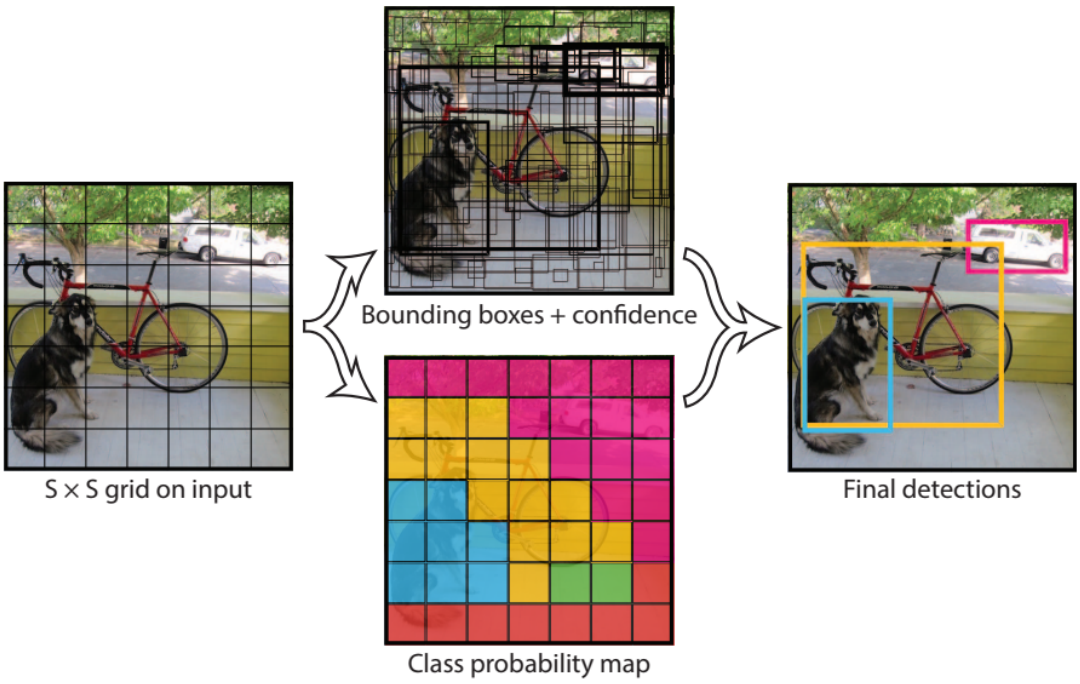


Abbildung 3.6: Vereinfachte Darstellung von Yolo Objekterkennung. Quelle: [14]

YOLO ist bedeutend schneller als andere Objekterkennungsalgorithmen. Hier liegen teilweise Unterschiede von über 40 Bildern pro Sekunde vor. Es ist schneller als Faster RCNN. Man muss jedoch sagen, dass es dafür um einen geringen Wert weniger genau die Objektklassen bestimmt. Schwierigkeiten hat YOLO außerdem mit dem Erkennen sehr kleiner Objekte.[14] Es gibt mittlerweile mehrere Nachfolger von YOLO, wie YOLOv2 (YOLO 9000) und YOLOv3 (die aktuellste Version). Diese zeigen eine Verbesserung in Leistung als auch Genauigkeit auf.YOLOv3 kommt in einer seiner Versionen auf bis zu 220 FPS. Außerdem lassen sich mit YOLOv3 unter anderem neue Objektklassen an ein bereits bestehendes Modell antrainieren. Joseph Chet Redmon, stellt YOLO so wie Umsetzungen für verschiedene Sprachen auf seiner Webseite[2] zur Verfügung.

[2] https://pjreddie.com/darknet/yolo/

3.1.3 Single Shot Detection

Auch der Single Shot Detection Algorithmus (SDD) nutzt nur das Eingangsbild, keine Seperaten Bildregionen. Für das Training werden auch hier Vergleichsboxen benötigt, die in den Trainingsbildern bereits richtig Klassifizierte und detektierte Objekte bestimmen. Beim SDD werden Feature Maps von unterschiedlichen Maßstäben (beispielsweise 8×8 oder 4×4) erzeugt, über die mehrere Standard- Boxen (beispielsweise 4) verschiedener Größe iterieren. Für jede dieser Standardboxen werden Box Offsets, sowie die Wahrscheinlichkeiten aller möglichen Objektklassen vorausgesagt. Während des Trainings wird versucht die Standardboxen an die vorher erstellten Vergleichsboxen anzupassen. Dabei werden nur die Standartboxen versucht anzupassen, die die gleiche Objektklasse sowie eine Wahrscheinlichkeit, die über einem Schwellenwert (standardmäßig 0.5) liegt, besitzen.[15] Abbildung 3.7 zeigt eine Visualisierung der sich auf dem Feature Maps bewegenden Boxen.

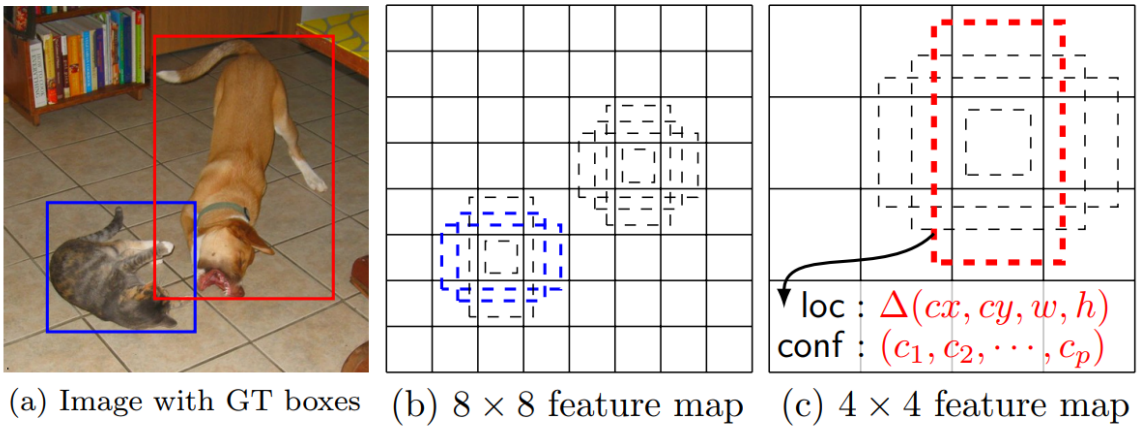


Abbildung 3.7: Vereinfachte Darstellung von SDD Multiboxen. Quelle: [15]

SDD ist genau so wie YOLO schneller als Faster R-CNN. Es gibt mehrere Testdurchläufe in denen SDD eine höhere Richtigkeit der Objektdetektionen, auf höher aufgelöste Input Bilder und einer höheren FPS Zahl erreicht.[15]. SDD wird demnach auf oft als Beispiel genannt, wenn es darum geht, Echtzeitobjekterkennung auf weniger Leistungsstarken Geräten lauffähig zu machen.

3.2 Beispiele der Sonifikation

Sonifikation ist die Verklanglichung verschiedenster Arten von Daten. In dieser Arbeit werden erkannte Objekte einer Umgebungssituation über 2D Bilder vertont. Dabei Bilden

wir Lautstärke auf Objektgröße und Entfernung ab. Außerdem soll die horizontale Position des Objekts durch unterschiedliche Lautstärkeverteilung auf Den rechten sowie linken Lautsprecher abgebildet werden.

Ein älteres Beispiel für eine Anwendung, die ebenfalls Objekte erkennt und ihre Position, so wie ihren Abstand zum Ausgangspunkt vertont ist Sonar (sound navigation and ranging). Beim Sonar handelt es sich um eine Unterwasser Objekterkennung über Schall. Schallwellen breiten sich unter Wasser schnell und annähernd verlustfrei aus. Es werden über horizontale Schallwellen Objekte ermittelt die selbst Schallwellen erzeugen. Dabei wird über die Dauer bis zum Aufeinandertreffen der Wellen die Entfernung berechnet und über die Aufprallposition die Richtung ermittelt. Die erkannten Objekte werden über die Frequenz eines abgespielten Tons wiedergegeben. je näher das Objekt, desto kürzer sind die Abstände zwischen den Tönen.[16] Da wir hier keine Bilddaten verwenden, sondern den Schallmessungen als Ausgangsgröße nehmen, lässt sich hier nur bedingt ein Vergleich ziehen. Jedoch haben wir ebenfalls erkannte Objekte, über die wir Informationen zu Entfernung auf ein akustisches Signal abbilden.

Mit dieser Arbeit soll außerdem gezeigt werden, dass es mit der Sonifikation von in Echtzeit erfassten Umgebungssituationen möglich ist, nur über das Gehör mit dem Objekt zu interagieren. Ein ähnlicher Zusammenhang wurde in der Studie "TANGIBLE ACTIVE OBJECTS AND INTERACTIVE SONIFICATION AS A SCATTER PLOT ALTERNATIVE FOR THE VISUALLY IMPAIRED"[17] untersucht. Hier wurde eine Umgebung erzeugt auf der sich Punktwolken aus Punkten, die verschieden wichtige Informationen zugewiesen bekommen haben, befanden. Sogenannte TAOs (greifbare aktive Objekte) konnten diese Punktwolke abtasten und die Wichtigkeit der Informationen eines Punktes auf die Tonhöhe eines Akustischen Signals abbilden. Je höher der Ton desto wichtiger der Punkt.

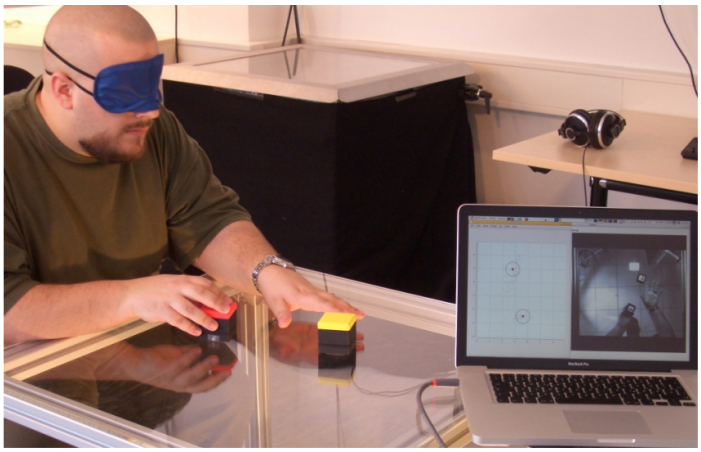


Abbildung 3.8: Testperson interagiert mit TOAs. Quelle: [17]

Durch ein Experiment in der eine Testperson diese TAOs über über ihre Punktwolken verschob, bis er die TAOs auf den wichtigsten Punkten abgelegt hat, wurde gezeigt, dass die Testperson rein akustisch Daten und Umgebungen analysieren konnte und in der Lage war die Objekte an der richtigen Stelle zu platzieren. Dadurch ist er in der Lage über Akustik und Handbewegungen beispielsweise Skatterplot Diagramme auszuwerten.

4 Training und Sonifikation

In diesem Kapitel wird der praktische Teil der Arbeit beschrieben. Bei der praktischen Arbeit ging es darum, eine eigene Objektklasse zu wählen und damit ein Modell zu trainieren. Dieses Modell soll in der Lage sein Objekte aus dieser Klasse in Echtzeit, mit möglichst hoher Richtigkeit der Klassifizierung zu erkennen und seine Position und Größe mit einer Bounding Box zu visualisieren. Im zweiten Schritt wird diese Objektklasse auf akustische Signale abgebildet.

4.1 Systemkonfiguration und Werkzeuge

Für die Implementierung der Aufgabenstellung wurde Python 3.6 Anaconda unter Windows 10 verwendet. Anaconda umfasst bereits viele kleinere Python Erweiterungen und Toolkits. Im Folgenden werden einige der hinzugefügten Hauptbibliotheken und Toolkits vorgestellt.

4.1.1 Tensorflow

Tensorflow (TF) ist eine 2015 ursprünglich von Google entwickelte open source Softwarebibliothek. Sie bietet vor allem Unterstützung in den Bereichen Machine Learning und Deep Learning. Tensorflow wurde in Python und C++ entwickelt und wird von Python Programmen genutzt. Mit Tensorflow lassen sich Neuronale Netze erstellen und trainieren.[18]

Zum Training des eigenen Modells bietet sich die GPU- Version von Tensorflow an. Voraussetzungen dafür sind:

- NVIDIA Garfikkarte mit aktuellen GPU Treibern
- CUDA Toolkit (NVIDIA Entwicklungsumgebung für GPU Anwendungen)
- cuDNN (Deep Neural Network Bibliothek für CUDA)

Die für diese Arbeit verwendeten Versionen waren Tensorflow GPU 1.9 und CUDA Toolkit v9.0.

4.1.2 Tensorflow Object Detection API

Die Tensorflow Object Detection API ist ein open source framework zum Konstruieren, Trainieren und Bereitstellen von Modellen zur Objekterkennung. Sie stellt eine Auswahl trainierbarer Modelle wie beispielsweise SDD with MobileNet und Faster R-CNN Inception Resnet v2 zur Verfügung.[19]

Zur Installation und Einrichtung wurde in dieser Arbeit die Github Bibliothek[1] von EdjeElektronics verwendet.

4.1.3 Open CV

Open CV ist eine umfangreiche open source Computer Vision Bibliothek. Die Bibliothek ist plattformunabhängig und in C und C++ geschrieben. Die Bibliothek besteht unter anderem aus mehr als 2500 optimierten Algorithmen.[20] Open CV bietet außerdem viele einfache Befehle die beispielsweise Videoskallierungen anpassen oder die Visualisierung von Kameraaufnahmen und Bildern ermöglichen.

4.1.4 Pygame

Pygame ist eine open source Python Programmiersprache zum erstellen von Multimediaanwendungen (hauptsächlich Spiele).[21] Pygame bietet vereinfachten Zugriff auf Tastatur, Maus und Audioausgabe.

Für diese Arbeit wurde Pygame ausschließlich für die Sonifikation genutzt. Besonders Hilfreich war dabei die Möglichkeit, mehrere parallel laufende Audiochannel erstellen und ansteuern zu können.

[1] https://github.com/EdjeElectronics

4.2 Erstellen von Trainingsdaten

Mit der Tensorflow Object Detection API lassen sich beliebige Objektklassen trainieren. Alle Trainingsdaten oder Trainingsbilder, die für die Erstellung des Modells erforderlich waren wurden selbst erhoben. Es war also eine Objektklasse von Nöten von denen die Möglichkeit bestand, ausreichend Objektbeispiele zu organisieren um Bilddaten davon zu erstellen. Demnach wurde als Objektklasse die Tasse (cup) gewählt. Die Trainingsdaten bestanden aus verschiedenen Fotos von Tassen. Die Umgebung der Tassen wechselte dabei von einfarbigen Hintergründen zu sehr unruhigen Hintergründen mit mehreren anderen Objekten. Außerdem waren Bilder dabei, die mehrere Tassen gleichzeitig zeigten und Überlappungen der Tassen untereinander beinhalteten.

(a) Einzelaufnahme **(b)** Erhöhte Anzahl **(c)** Erhöhte Entfernung

Abbildung 4.1: Beispielbilder aus dem Trainingsbilder- Datensatz.

Insgesamt wurden beinhaltete der Datensatz 201 Bilder. Um die Trainingsdauer zu reduzieren wurden alle Bilder auf eine Auflösung von 800×600 runter gerechnet. Um die Bilder auf das Training vorzubereiten wurden 19 der 201 Bilder (ca. 10 Prozent) in einen Testordner verschoben. Die restlichen Bilder befanden sich im Trainingsordner. Die Testbilder sind nötig gewesen, um die Abweichungen der Vorhersagen zu minimieren und das bisher Trainierte Modell zu nach jedem Trainingsschritt zu testen. Als letztes mussten die Bilder noch gelabelt werden. Dazu wurde labelImg verwendet. Ein weiteres open source Tool um label Dateien für manuell erstellte Bounding Boxen zu generieren. Am Ende muss für jede Bilddatei sowohl im Test, als auch im Trainingsordner eine zugehörige Labeldatei (XML- Dateien) vorhanden sein. Als nächstes müssen die Labeldateien über ein Pythonscript (tf record) umgewandelt werden, sodass sie beim Training zum Klassifizieren genutzt werden können. Außerdem wird noch eine Labelmap Datei benötigt. Diese fiel in unserem Beispiel eher simpel aus, da wir nur ein Label (cup) hatten. Für die Modelle die zum trainieren genutzt wurden musste dann noch die Pfade in der configdatei auf die vorher erstellten Dateien geändert werden.

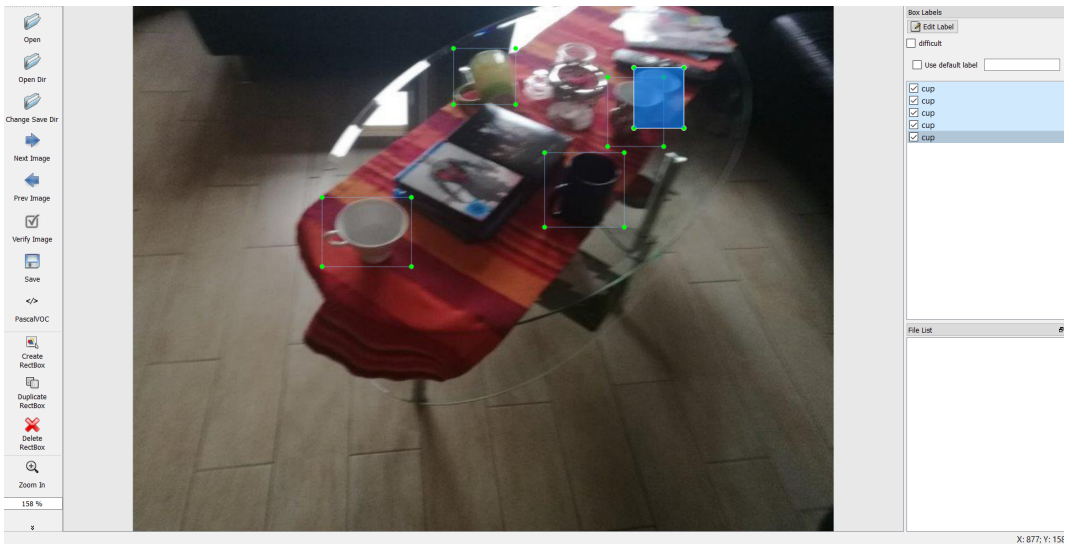


Abbildung 4.2: Labeln von Bilddaten mit Hilfe von labelImg

4.3 Trainingsverlauf und Ergebnis

Für das Training wurde als erstes das von Google bereitgestellte Modell Faster R-CNN inception v2 COCO verwendet. Der COCO (Common Objects in Context) steht hier für den Datensatz mit dem das Modell trainiert wurde. Dieses Modell trainierte den Cup- Datensatz mit 16205 Durchläufen. Jeder Durchlauf dauerte im Durchschnitt 4 Sekunden. Das entspricht einer Trainingsdauer von rund 18 Stunden. Das Training wurde manuell beendet nachdem der durchschnittliche Loss- Wert unter 0.05 war.

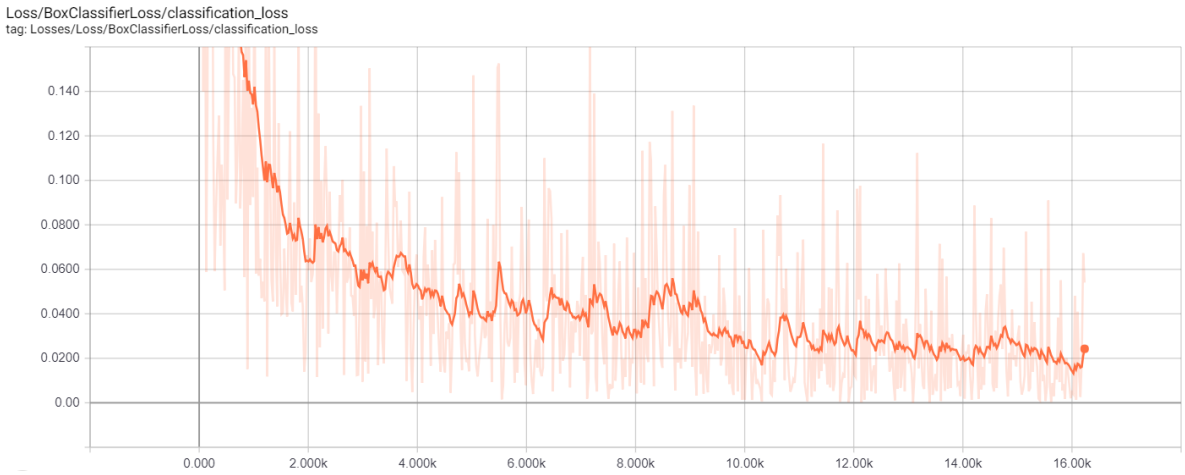


Abbildung 4.3: Tensorboar Loss Graph Faster R-CNN

Nachdem das Training abgeschlossen war, wurde das Modell in einer Webcam Situation getestet. Die Objekterkennung an sich hat gute Ergebnisse erzielt. Es trat je-

doch ein signifikantes Problem auf. Durch die niedrige Rechenleistung des PCs der für die Objekterkennung genutzt wurde, gab es in der Live Aufnahme nur alle drei Sekunden ein neues Bild. Es ließ sich also nicht von Echtzeitobjekterkennung reden. Demnach musste eine Variante her, die auch auf dem zur Testrechner lief der zur Verfügung stand.

Das Training wurde neu gestartet mit dem selben Tassen Datensatz wie vorher. Das Modell wurde jedoch von SSD mobilenet v1 COCO ersetzt. Dieses sollte nicht ganz so korrekte Ergebnisse erzielen, sich aber dafür durch eine höhere FPS Zahl auszeichnen. Die Trainingsdauer der SSD- Variante dauerte bedeutend länger. Ein Trainingsdurchlauf dauerte rund 21 Sekunden. Nach 8 Stunden Training war der Loss Wert zwar nicht konstant bei unter 0.05, eine starke Verbesserungskurve war trotzdem zu erkennen. Das Training wurde an der Stelle wieder manuell beendet und Das Modell in einer Webcamsituation getestet. Die Bildrate war sehr gut und die Funktionalität zufriedenstellend. Tassen wurden konstant in verschiedenen Winkeln erkannt. Der Treshhold der angezeigten Bounding Boxen musste jedoch von 0.8 auf 0.6 gesetzt werden. Das Modell erkennt auch Tassen ähnliche Formen ab und an als Tasse an. Das ist jedoch bei den wenigen Trainingsdurchläufen und der kleinen Menge an Testbildern normal. Nichtsdestotrotz lieferte das Modell Ergebnisse die zum vertonen reichten.

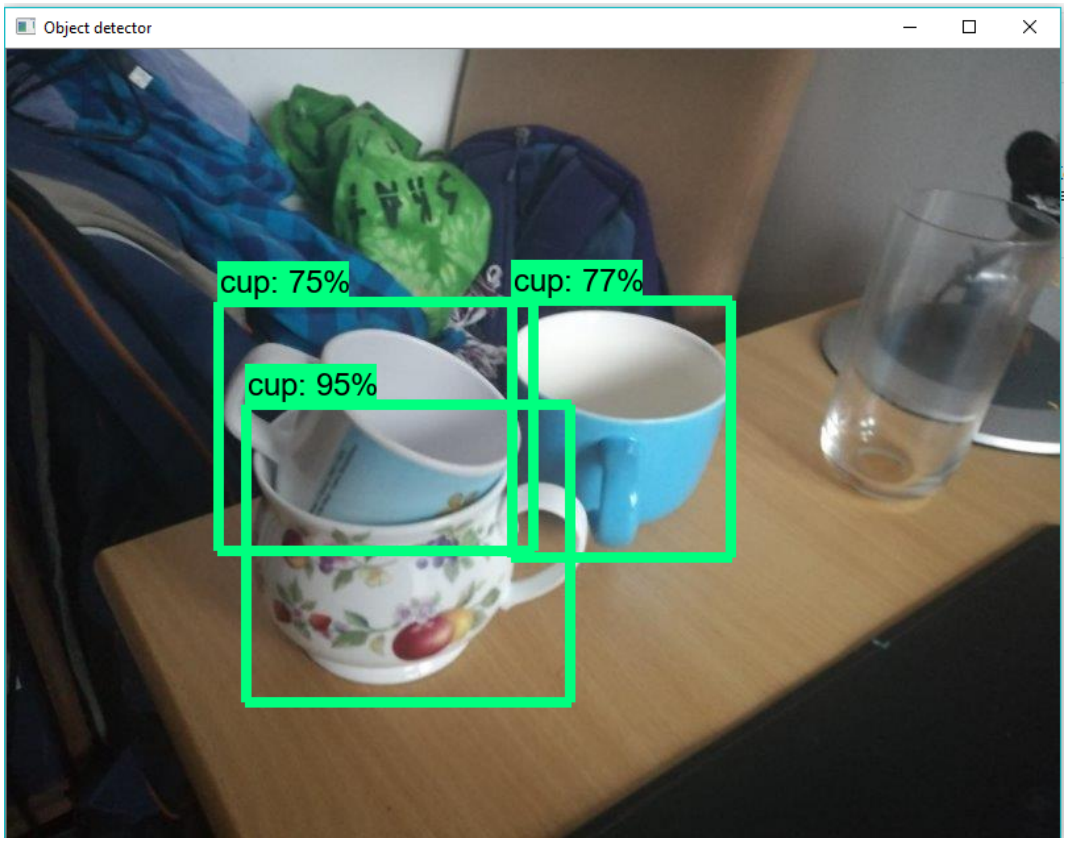


Abbildung 4.4: Tassen Erkennung mit SSD Modell

4.4 Vertonung des Modells

Das Konzept der Vertonung orientiert sich an verschiedenen Objekteigenschaften im Bild. In erster Linie sollte überhaupt ein Ton zu hören sein, wenn das Objekt erkannt wird. Dieser Ton sollte in Abhängigkeit zur Größe der erkannten Bounding Box seine Lautstärke anpassen. Je näher das Objekt, umso größer die Bounding Box und umso lauter der abgespielte Ton. Es war also Notwendig an den Flächeninhalt der Bounding Boxen zu gelangen. Die TF Object Detection API gibt ein Array mit allen erstellten Boxen aus. Das erste Element in diesem array ist das mit dem höchsten Score. Um an die aktiven Boxen zu gelangen iteriert man also von der ersten Stelle des Arrays an durch, bis die erste Box kommt, dessen Score unter dem Schwellenwert liegt. Für diese aktiven Boxen sind normierte Minimum und Maximum Werte für x und y bekannt. Diese werden mit der Bildbreite- bzw. Höhe multipliziert. Dadurch bekommt man die Pixelwerte mit deren Hilfe man den Flächeninhalt berechnen kann. Mit Hilfe von Pygame wurde dann der Flächeninhalt in Relation zur Lautstärke des Tons berechnet. eine Lautstärke von 100 Prozent ist erreicht wenn die Bounding Box größer gleich der Bildschirmgröße ist.

Ein weiteres Ziel war es die Richtung, aus der das Objekt kommt, bestimmen zu können. Pygame hat die Funktion die Lautstärke prozentual auf links und rechts aufzuteilen. Anhand der Breite der Bounding Box und ihrem Verhältnis zur Vertikalen Mittellinie des Bildes wurde berechnet zu welchen Anteilen sie sich links bzw. rechts befindet. Im Anschluss passt man die vorher berechnete Lautstärke an die Seitenverhältnisse an und gibt den Ton entsprechend aufgeteilt aus.

Die letzte Größe die erkannt werden sollte, ist die Anzahl der Objekte. Wenn mehrere Tassen gleichzeitig erkannt werden, so macht es keinen Sinn ihnen den gleichen Ton für beide abzuspielen. dieser wird dann einfach nur Lauter und man kann nicht unterscheiden ob es zwei entfernte Objekte sind oder ein Objekt, was sich nah an der Kamera befindet. Demnach wurde, immer wenn eine weitere Tasse zu den aktiven Tassen hinzukam, ein neuer separierbarer Ton abgespielt. In der Grundlautstärke waren die Töne alle gleich. Für die später durchgeführten Tests haben drei verschiedene Töne gereicht. Demnach wurden auch nicht mehr implementiert. Auf der nachfolgenden Seite befindet sich der Quellcode zur Vertonung.

Sonifikation von Bounding Boxen

```
i = 0
is_displayed = True
active_boxes = 0

while(is_displayed):

    if (float(scores[0][i])>=min_score):

        ymin = int((boxes[0][i][0]*height))
        xmin = int((boxes[0][i][1]*width))
        ymax = int((boxes[0][i][2]*height))
        xmax = int((boxes[0][i][3]*width))
        box_area = int((ymax - ymin)*(xmax - xmin))
        image_mid = width / 2

        if (xmax < image_mid):
            left_part =  (1.0 / image_mid * (xmax - xmin))
            right_part = 0.0

        if (xmin >image_mid):
            right_part =  (1.0 / image_mid * (xmax - xmin))
            left_part = 0.0

        if ((xmin < image_mid) & (xmax > image_mid)):
            right_part = (1.0 / image_mid * (xmax - image_mid))
            left_part = (1.0 / image_mid * (image_mid - xmin))

        if (float(classes[0][i]) == 1.0):

            print("Tasse")
            print(active_boxes)

            #volume calculatd by bounding box size, 100% of the picture = 100% volume
            cup_sound_volume = (1.0 / (height*width)) * box_area

            cup_channel = pygame.mixer.Channel(active_boxes)
            cup_channel.play(cup_sound_array[active_boxes], loops = -1)
            #seperate volume to right and left speaker by their percentages
            cup_channel.set_volume((left_part * cup_sound_volume),(right_part * ↩
                cup_sound_volume))

            if (active_boxes<2):
                active_boxes+=1
                print(active_boxes)

        i+=1
        Result = [ymin,ymax,xmin,xmax]
        print(Result)
        print(box_area)

    else: is_displayed = False
```

5 Evaluierung

In diesem Abschnitt geht es abschließend darum, das trainierte Modell und seine Sonifikation auf Interaktionsmöglichkeiten zu testen. Dazu wurden drei Versuche durchgeführt, dessen Ergebnisse vorgestellt und diskutiert werden.

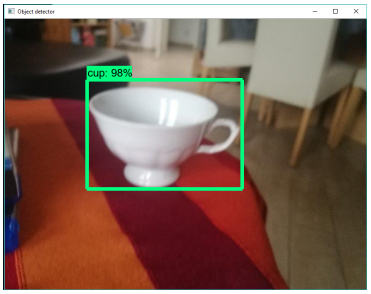
5.1 Experiment: Blinde Objektinteraktion

Alle folgenden Versuche wurden hintereinander mit drei verschiedenen Probanden durchgeführt. Jeder Proband hat kurz vor dem Versuch erfahren, was seine Aufgabe sein wird. Es gab also keine Zeit sich vorzubereiten. Während jedem der Versuche waren die Probanden blind. Sie konnten weder sehen, was auf dem Bildschirm passiert, noch ihre optische Umgebung wahrnehmen.

5.1.1 Versuch 1: Wahrnehmen von Grundeigenschaften

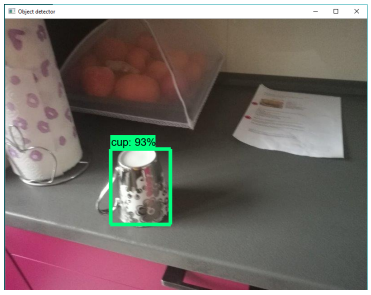
Beim ersten Versuch wurden nacheinander Bilder von sieben Tassen aufgerufen. Die Probanden waren nicht in der Lage diese Bilder zu sehen. Die einzigen Informationen die sie bekommen haben waren die Töne, die die Objekterkennung wiedergegeben hat. Zur Orientierung wurde ihnen beim ersten Bild gesagt wo sich die Tasse befindet, damit sie einen ungefähren Vergleichswert haben. Ab dem zweiten Bild wurden sie dann nach der Anzahl der Tassen, ihrer Entfernung und horizontalen Ausrichtung befragt (eher rechts, links oder mittig).

(a) Bild 1



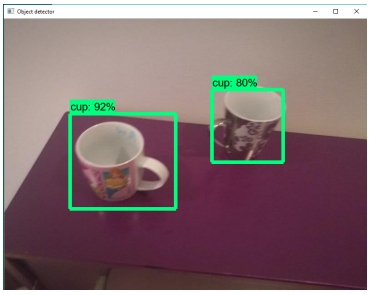
(b) Bild 2

(c) Bild 3

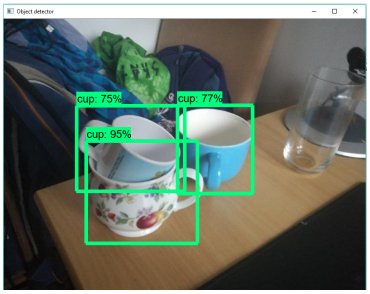


(d) Bild 4

(e) Bild 5



(f) Bild 6



(g) Bild 7

Abbildung 5.1: Versuchsbilder, Versuch 1

Da die meisten Eigenschaften richtig erkannt wurden, werden nur Fehler ausgewertet. Bild 3 war das erste Bild was Schwierigkeiten bereitet hat. Testperson 1 und 2 haben in Bild 3 mehr als eine Tasse wahrgenommen. Der nächste Fehlerschwerpunkt war Bild 6. Die Anzahl der Tassen wurde von allen richtig ermittelt, jedoch hat jeder Probant die rechte Tasse als näher an der Kamera bewertet. Bild 7 wurde von Probant 1 komplett richtig bewertet. Probant 2 hat rechts 2 Tassen angenommen und links eine. Probant 3

war sich sicher, dass es zwei Tassen sind. Wie könnten die falschen Einschätzungen Zustande kommen? Die folgenden Aussagen ergaben sich aus Gesprächen mit den Probanden.

Bild 3 besitzt die die größte Bounding Box. Der wiedergegebene Ton war an der Stelle sehr laut. Bei dieser Lautstärke kam Unsicherheit auf, da bei der erhöhten Lautstärke der Ton an sich anders wahrgenommen wird. Bei Bild 6 kam das erste mal eine zweite Tasse hinzu. Die Lautstärke des zweiten Geräusches ist die gleiche wie die des ersten. Jedoch ist der Ton ein anderer. Unterschiedliche Tonhöhen bzw. Töne können auch bei gleicher Lautstärke unterschiedlich wahrgenommen werden. Vor allem dann, wenn der Ton das erste mal gehört wird und kein Vergleichswert zu diesem Ton existiert. Bild 7 war sehr schwer einzuordnen, da sich alle Tassen in der Mitte häufen. Demnach ist die rechts-links Unterteilung fehlerhaft.

Abbildung 5.2: Testperson bei der blinden Ermittlung von Positionseigenschaften

5.1.2 Versuch 2: Interaktion mit bewegten Bildern

Bei Versuch zwei haben die Probanden ihre sitzende Position beibehalten. Anstelle eines Bildes lief jetzt ein Video, indem abwechselnd von den Seiten Tassen näher an die Kamera kamen. Sie mussten eine ausweichende Bewegung simulieren, wenn sie der Annahme waren von ein Objekt kommt ihnen zu nah. Sie mussten also sowohl auf Lautstärke reagieren, als auch auf die Richtung aus der das Objekt auf sie zu kamen.

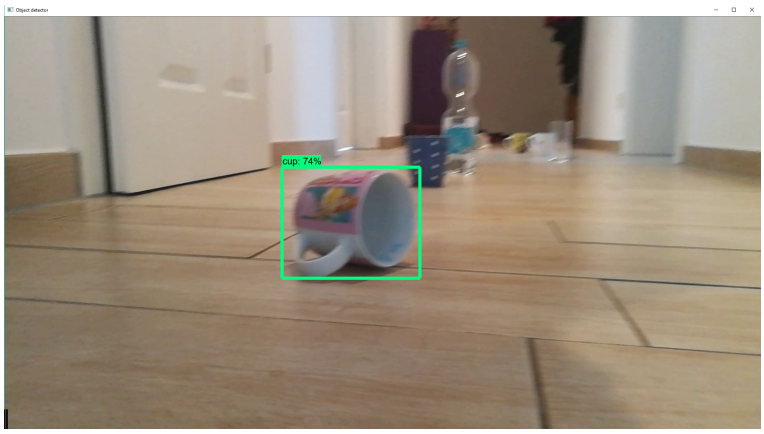


Abbildung 5.3: Videoauschnitt 1

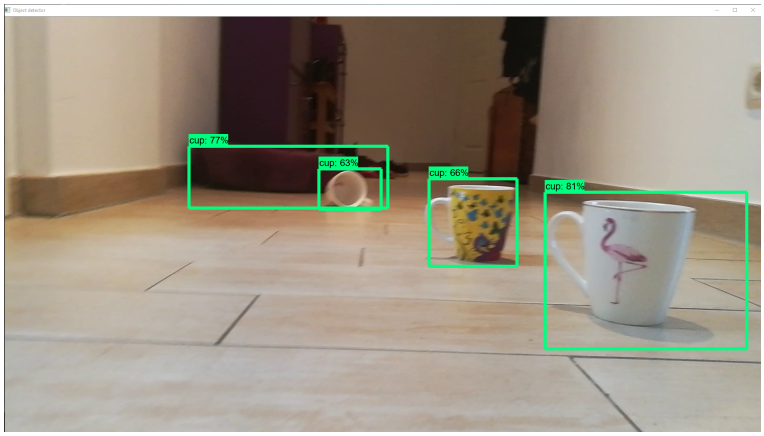


Abbildung 5.4: Videoauschnitt 2

Schwierigkeit hatten die Probanden hier am Ende des Videos. Einfache Ausweichszenarien waren kein Problem. Auch für kurze Zeit falsch Detektierte Objekte wurden als nicht konstant und unwichtig wahrgenommen. Auf Abbildung 5.4 sieht man jedoch, dass das Katzenbett ganz im Hintergrund auch als Tasse bestimmt wird. Dieses ist größer als die Tassen im Vordergrund und wird deshalb auch lauter dargestellt. die Probanden sind alle mehrfach nach rechts ausgewichen, weil ihnen vorgetäuscht wurde, dass dort eine Tasse im Vordergrund erkannt wurde.

5.1.3 Versuch 3: Objektsuche

Beim letzten Versuch hielten die Probanden eine Webcam auf Hüfthöhe in der Hand. vor ihnen befanden sich in einer horizontalen Reihe nebeneinander angeordnet 4 verschiedene Objekte. Eins dieser Objekte war eine Tasse. Sie durften sich beliebig oft von links nach rechts bewegen, bis sie sich sicher waren, dass sie vor der Tasse stehen.

Abbildung 5.5: Webcam Objektsuche

Die anderen Objekte wurden zeitweise auch als Tasse klassifiziert. Da die Probanden aber alle schon ungefähr abschätzen konnten, wie konstant das Geräusch sein muss, damit es sich beim Objekt wirklich um eine Tasse handelt, haben alle die Tasse im ersten Versuch gefunden. Bei Zylinderförmigen Objekten, wie dem Sockel der Holzpuppe zögerten alle. Dieser wurde auch sehr wahrscheinlich und kontinuierlich als Tasse erkannt.

5.2 Interpretation der Ergebnisse

Grundlegen lässt sich durch die Experimente sagen, dass alle Aufgaben gut bis sehr gut bewältigt werden konnten. Versuch 1 hat gezeigt, dass es schwierig wird Situationen zu erkennen, sobald sich die Objekte häufen und auf einer Höhe liegen. Außerdem, wurden die Lautstärken bei mehreren Tönen gleichzeitig nicht immer richtig eingeschätzt. Daraus lässt sich folgern, dass die Töne ein signifikant unterschiedliches Klangbild haben sollten damit klar ist, wie viele Objekte sich gleichzeitig vor einem befinden. Versuch 2 hat gezeigt, dass die Abbildung der Lautstärke auf Objektgröße über den Flächeninhalt der Bounding Box nur so lang funktioniert, wie die Objekte einer Klasse alle nah an ihrer Durchschnittsgröße liegen. Fällt ein Objekt aus der Rolle so kann das schnell zu Täuschungen führen. Versuch 3 hat gezeigt, dass das Modell noch mehr Daen und noch mehr Training gebraucht hätte. Wenn die Vorhersagen noch eindeutiger gewesen wären, hätten die Probanden ga nicht gezweifelt.

Was jedoch der Mehrwert von Sonifikation von Objekten ist, war deutlich zu erkennen. Alle Aufgaben konnten von den Probanden gemeistert werden. Jeder war in der Lage, mit einer kurzen Einführung, mit den erkannten Objekten nur über sein Gehör zu interagieren. Außerdem war eine Lernkurve festzustellen. Die Probanden wurden von Aufgabe zu Aufgabe sensibler für die jeweilige Vertonung. Sie haben am Ende viel präziser wahrgenommen als anfangs. Würde man das Konzept ausbauen und das Modell noch zuverlässiger machen. Dann wäre mit wenig Aufwand eine zuverlässige Vertonung möglich.

6 Fazit

Die Testergebnisse des Modells und seiner Sonifikation haben gezeigt, dass Probanden in der Lage sind rein akustisch mit diesem Modell zu interagieren. Informationen über Position, Entfernung, Häufigkeit und Objektklasse ließen sich durch akustische Reize wahrnehmen und erlernen. Die Genauigkeit des Modells ist als ausbaufähig einzuschätzen. Mehr Trainingsdurchläufe und eine größere Menge an Testdaten, hätte die Versuchsergebnisse noch besser ausfallen lassen. Das Hauptproblem lag noch in der Fehldetektion von Objekten die nicht zur Objektklasse gehörten. Ein weiterer Ausblick lässt sich in Zusammenhang mit der Sonifikation ziehen. Selbst diese Rudimentäre Fassung der Vertonung war ausreichend um gute Versuchsergebnisse zu erzielen. Die Vertonung über die Bounding Box Fläche ist aktuell noch linear. Das könnte man ändern um es noch mehr an Echte akustische Wahrnehmung anzunähern. Außerdem macht es Sinn sich noch mehr mit der Auswahl der verschiedenen Töne zu beschäftigen. Der einzige Maßstab war bisher, dass diese gut Hörbar sind und man sie irgendwie unterscheiden kann.

Literaturverzeichnis

[1] W. BURGER and M.J. BURGE. *Principles of Digital Image Processing: Fundamental Techniques*. Undergraduate Topics in Computer Science. Springer London, 2010.

[2] W. KONEN and T. ZIELKE. Bildverarbeitung und Algorithmen. 2006. http://www.gm.fh-koeln.de/~konen/WPF-BV/BV06a.PDF , Abgerufen Dezember 2018.

[3] L. DENG and D. YU. Deep Learning: Methods and Applications. *Foundations and Trends® in Signal Processing*, 7(3–4), 2014.

[4] I. GOODFELLOW, Y. BENIGIO, and COURVILLE A. *Deep Learning*. MIT Press, 2016.

[5] M. NIELSEN. *Neural Networks and Deep Learning*. 2018.

[6] R. SATHYA and A. ABRAHAM. Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *(IJARAI) International Journal of Advanced Research in Artificial Intelligence, Vol. 2, No. 2*, 2013.

[7] P. SERMANET, D. EIGEN, X. ZHANG, M. MATHIEU, R. FERGUS, and Y. LECUN. Integrated Recognition, Localization and Detection using Convolutional Networks. 2014.

[8] DEEPLIZARD. Machine Learning & Deep Learning fundamentals. 2017. http://deeplizard.com/learn/video/YRhxdVk_sIs, Abgerufen Dezember 2018.

[9] Y. ZENG, X. XU, Y. FANG, and K. ZHAO. Traffic Sign Recognition Using Extreme Learning Classifier with Deep Convolutional Features. *The 2015 international conference on intelligence science and big data engineering (IScIDE 2015)*, 2015.

[10] J.R.R. UIJILINGS, K.E.A. VAN DE SANDE, T. GEVERS, and SMEULDERS A.W.M. Selective Search for Object Recognition. *IJCV*, 2012.

[11] S. REN, K. HE, R. GIRSHIK, and J. SUN. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*, 2016.

[12] F. LI, J. JOHNSON, and S. YEUNG. Detection and Segmentation. 2017. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf, Abgerufen: Dezember 2018.

[13] K. HE. Convolutional Feature Maps, Elements of efficient (and accurate) CNN-based object detection. *Microsoft Research Asia*, 2015. http://kaiminghe.com/iccv15tutorial/iccv2015_tutorial_convolutional_feature_maps_kaiminghe.pdf, Abgerufen Dezember 2018.

[14] J. REDMON, S. DIVVALA, R. GIRSHICK, and A. FARHADI. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[15] W. LIU, D. ANGUELOV, D. ERHAN, C. SZEGEDY, S.E. REED, C. Y. FU, and A. C. BERG. SSD: single shot multibox detector. *CoRR*, 2015.

[16] A. A. WINDER. Sonar System Technology. *IEEE Transactions on Sonics and Ultrasonics*, 1975.

[17] E. RIEDENKLAU, T. HERMANN, and H. RITTER. Tangible Active Objects and interactive sonification as a scatter plot alternative for the visually impaired. *The 16th International Conference on Auditory Display (ICAD-2010)*, 2010.

[18] TENSORFLOW About. URL: https://www.tensorflow.org/.

[19] J. HUANG, V. RATHOD, C. SUN, M. ZHU, A. KORATTIKARA, FATHI A., I. FISCHER, WOJNA Z., Y. SONG, S. GUADARRAMA, and K. MURPHY. Tensorflow Object Detection api. *CVPR*, 2017. GITHUB. URL: https://github.com/tensorflow/models/tree/master/research/object_detection.

[20] OPEN CV About. https://opencv.org/about.html.

[21] PYGAME About. https://www.pygame.org/wiki/about.