Universität Bremen

Fachbereich 3: Mathematik und Informatik

Bachelorarbeit

Verbesserung eines Ballerkenners mittels hocheffizienten CNN mit synthetischen Trainingsdaten

Arne Neisser
Matrikelnummer: 4434156

19.10.2020

1. Gutachter: Prof. Dr.-Ing. Udo Frese
2. Gutachter: Prof. Dr. Rolf Drechsler

Erklärung

Ich versichere, die Bachelorarbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilftmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Bremen, den 19.10.2020

_____

Arne Neisser

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In der aktuellen Forschung um maschinell lernende Systeme scheint es, dass mehr relevante Trainingsdaten eine Verbesserung der Erkennungsleistung des Systems erwirken [7, 17]. Daten für das Training solcher Systeme stammen entweder aus dem jeweiligen Feld oder werden aus ähnlichen Aufgaben übernommen und weiter verarbeitet. Um ein Training im überwachten Lernen zu erreichen, muss die zu trainierende Leistung zuerst anderweitig erbracht werden. Erst dann kann diese dem lernenden System beigebracht werden. Das Aufnehmen und Annotieren von Trainingsdaten sind bislang stark einschränkende Faktoren bei dem Training lernender Systeme. Durch einen automatisierten Prozess zum sinnhaften Vergrößern der Menge der Trainingsdaten mit automatisch generierter Annotation könnte somit trotz weniger realer Trainingsdaten von einem sehr speziellen Feld gelernt werden.

1.2 Zielsetzung

Im Rahmen dieser Arbeit soll ein bestehendes Erkennungssystem in der Genauigkeit verbessert werden. Ein neuronales Netz soll eine hoch-performante 2D-Objektlokalisierung auf Kamerabildern leisten und eine Bildmaske generieren, welche die Suchbereiche des bestehenden Systems einschränkt. Dafür soll das Netz von computergenerierten Bildern die Lokalisierung von Bällen lernen und später eine schnelle und genaue Erkennung im laufenden System leisten. Die Implementierung und Evaluierung des Verfahrens findet am Roboter *Doggy* der AG Multisensorische Interaktive Systeme statt und soll dort verstärkend Falscherkennungen des Mittelpunktes vermeiden. Insgesamt soll dies zu einer höheren Genauigkeit in der Trackingleistung der Bälle führen.

1.3 Aufbau der Arbeit

Im zweiten Kapitel wird der vorhandene Roboter und grundlegende Technologien für die Umsetzung vorgestellt. Danach werden in Kapitel 3 verwandte Arbeiten und ihre Ergebnisse erläutert. In Kapitel 4 wird die Synthetisierung von Trainingsdaten zusammen mit dem erarbeiteten System vorgestellt und im fünften Kapitel ist die Integration im bestehenden System beschrieben. In Kapitel 6 geht es um die Evaluierung der Entwicklung. Zuletzt wird ein Fazit über die Arbeit gezogen und ein Ausblick in die Zukunft gegeben.

Kapitel 2

Grundlagen

In diesem Kapitel werden die grundlegenden Technologien und Hintergründe dieser Arbeit erklärt. Convolutional Neural Networks (CNN), das MobileNetV2 als auch der vorhandene Roboter werden in folgenden Absätzen erläutert.

2.1 Convolutional Neural Networks

Convolutional Neural Networks sind neuronale Netzwerke, die besondere Arten von Schichten nutzen. Diese heißen Convolutional Schichten und basieren auf der Nutzung lokaler Zusammenhänge in den Eingabedaten. Die Eingabedaten sind meist mehrdimensionale Vektoren, die innerhalb des Netzes durch verschiedene Schichten zur Merkmalsextraktion laufen. Der erste Teil eines CNN sieht von der Struktur oft ähnlich aus und dient zur Extraktion von Merkmalen innerhalb des Bildes. Mit steigender Tiefe des Netzes wird das rezeptive Feld der Neuronen auf tiefen Schichten größer. Das liegt an der Struktur und der Idee der neu eingeführten Schichten, den Convolutional Schichten. Diese führen eine Faltung von ganzen lokalen Bereichen der Eingabe auf ein einzelnes Neuron der Ausgabe aus. Werden diese Faltungsschichten mit einer jeweiligen Kernelgröße von k wiederholt hintereinander verkettet, ergibt sich für ein spätes Neuron der i-ten Schicht ein rezeptives Feld von $(k^i \times k^i)$ auf dem Eingabevektor. Das erste neuronale Netz welches auf diesen Faltungen aufbaute war das LeNet-5 von LeCun et al. [6].

2.1.1 Convolutional Schicht

Convolutional- oder faltende Schichten machen sich die lokalen Zusammenhänge von Daten zu Nutze. Sie gehen davon aus, dass ein Wert aus einem mehrdimensionalen Vektor in seinen Informationen enger mit einem Nachbarwert zusammenhängt, als einer der am anderen Ende des

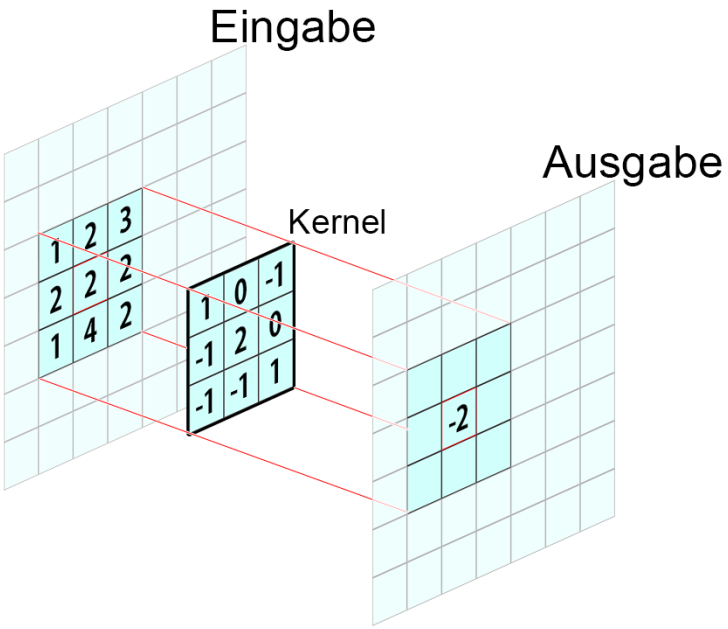

Abbildung 2.1 Visualisierung der Verrechnung eines Eingabebereiches mit einem k=3 Kernel zu einem Ausgabewert.

Vektors steht. Um diese lokalen Informationszusammenhänge zu nutzen werden in Convolutional Schichten Kernel bestimmter Größe verwendet. Ein Kernel wird durch eine Kernelmatrix beschrieben die in sich Faktoren, und somit Gewichtungen der Eingabewerte, trägt. Dieser wird bei der Berechnung eines Ausgabekanals über die Eingabekanäle wie ein Fenster geschoben und verrechnet alle, in den Kernel fallenden, Werte mit der Gewichtung aus dem Kernel auf einen Wert in der Ausgabe. In Abbildung 2.1 ist diese Faltung der Eingabe auf einen Wert der Ausgabe visualisiert. Die lernenden Parameter einer Convolutional Schicht liegen somit in den Kernel. Neuronen auf diesen Schichten bekommen einen kleinen, durch die Kernelmatrix eingeschränkten, lokalen Bereich von Informationen der vorherigen Schicht. Gleichzeitig gewichtet die Kernelmatrix diese Informationen, sodass eine Faltung dieser auf ein Neuron durchgeführt wird. Convolutional Schichten bestehen meist aus mehreren Kanälen, welche alle die gleichen Eingabeinformationen bekommen, jedoch verschiedene Kernelmatrizen besitzen, sodass verschiedene Merkmale pro Schicht (Feature Maps) entstehen können. Bei Convolutional Schichten mit einer Eingabedimension von $h_i \times w_i \times d_i$ und einer Ausgabedimension von $h_j \times w_j \times d_j$ mit einem Kernel $K \in \mathbb{R}^{k \times k \times d_i \times d_j}, k \in \mathbb{N}^+$ beläuft sich der Rechenaufwand auf $h_i \cdot w_i \cdot d_j \cdot k \cdot k$.

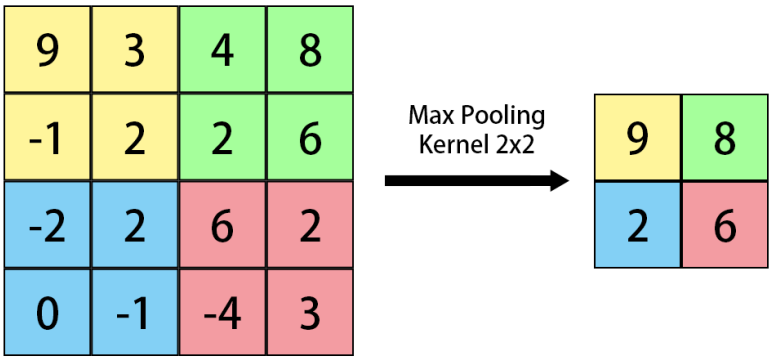

Abbildung 2.2 Visualisierung eines Max Pooling mit Kernelgröße 2x2.

2.1.2 Pooling Schicht

Eine Pooling Schicht dient zur Dimensionsverringerung ihrer Eingabedaten und somit Aggregation der Informationen. Grundsätzlich gibt es verschiedene Arten von Pooling Schichten die sich lediglich in der Berechnung ihrer Ausgabe unterscheiden. Alle besitzen jedoch einen Kernel, der einen Eingabebereich auf einen einzigen Wert reduziert. Wichtig dabei ist, dass diese Kernel sich nicht überlappen und somit jeder Wert nur einmal eingerechnet werden kann, anders beim Convolutional-Kernel. Die geläufigste Form ist das MaxPooling, bei dem nur der höchste Wert innerhalb des Kernels ausgewählt wird (Siehe Abbildung 2.2). Somit wird die stärkste Information der letzten Aktivierung weitergegeben. Eine andere Form ist das Average-Pooling bei dem der Durchschnitt aller Werte im Kernel gebildet wird und dieser als Ausgabewert fungiert. Zudem gibt es das Global Pooling, auf ähnlichem Prinzip und ähnlicher Idee beruhend, welches ebenfalls zur Aggregation von Informationen und zur Dimensionsverringerung dient, jedoch einen gesamten Kanal (Feature Map) auf einen einzigen Wert abbildet. Zur Berechnung dienen die gleichen Methoden wie beim normalen Pooling. Globales Pooling ist bei Klassifikationsaufgaben eines Netzes sinnvoll und kann die merkmalselektierenden Fully-Connected Schichten unterstützen, indem sie die Inhalte der Feature Map vereinfachen.

2.2 MobileNetV2

Das MobileNetV2 ist eine ressourcensparende Netzarchitektur die den Einsatz von CNN auf hardware-schwachen Geräten ermöglichen soll. Das MobileNetV2 führt zur Komplexitätsreduktion normaler Convolutional Schichten einen *inverted residual* Block ein. Dieser besteht zur Eingabe aus einer Expansionschicht, einer merkmalsselektierenden *Depthwise Separable Convolution*-Schicht in der Mitte und einer Engpass-Convolution zur Ausgabe.

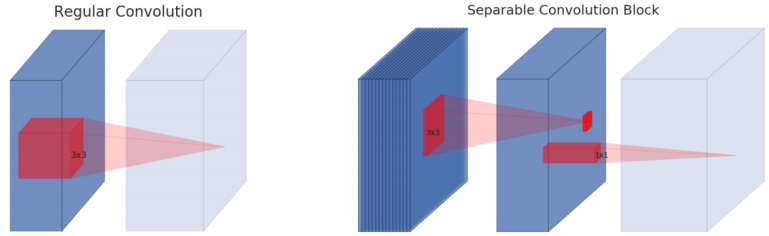

Abbildung 2.3 Normale Convolution und Seperable Depthwise Convolution

Modell	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Tabelle 2.1 Depthwise Separable- gegen konventionelles Convolution MobileNet [4]

2.2.1 Inverted Residual

Der *inverted residual* Block benutzt zur Reduzierung des Rechenaufwands *Depthwise Separable Convolutions*, welche aus zwei faktorisierten einzelnen Convolutionen bestehen, einer kanalweisen Convolution, welche einen einzelnen Filter pro Eingabekanal anwendet, und einer punktuellen, normalen 1×1 Convolution, die durch lineares kombinieren neue Merkmale aus der Eingabe bildet. Bei gleicher Ein- und Ausgabedimension beläuft sich der Rechenaufwand auf $h_i \cdot w_i \cdot d_i(k^2 + d_j)$ und reduziert diesen gegenüber einer normalen Convolutional Schicht somit fast um den Factor k^2. [11].

Zu Beginn eines *inverted residual* Blocks expandiert eine normale Convolution mit 1×1 Kernelgröße die Kanalzahl um einen Expansionsfaktor t, um einen größeren Raum für Merkmale zu erhalten. Die Depthwise Convolution bildet ein Merkmalskanal pro Kanal und durch die punktuelle 1×1 Convolution werden diese getrennten Merkmale verknüpft. Die Idee der anschließenden Engpass-Convolution ist, dass in dieser die wichtigsten gefundenen Merkmale selektiert werden.

Während der Erarbeitung des traditionellen MobileNet wurde ein Experiment durchgeführt um die Vorteile der Konstruktion der *Depthwise Separable Convolutions* aufzuzeigen. Eine Architektur mit normalen Convolutional Schichten wurde gegen eine Architektur mit *Depthwise Separable Convolutions* auf dem ImageNet Datensatz trainiert und evaluiert. Mit 14,3% der Parameter der normalen Convolutional Architektur, erzielte die neu erarbeitete Architektur ein nur um 1,1% schlechteres Accuracy Ergebnis. Siehe Tabelle 2.1.

Abbildung 2.4 Namensgebenes Kostüm für den Roboter Doggy.

2.3 Ballspielsystem Doggy

Der Prototyp des ballspielenden Unterhaltungsroboter Doggy (Abbildung 2.4) wurde innerhalb einer Arbeit von Tim Laue et al. [5] zur kommerziellen Nutzbarkeit von Unterhaltungsrobotern erarbeitet. Die Sportrobotik ist ein Feld, welches von dem jährlich stattfindenden RoboCup Wettbewerb geprägt ist. Allerdings geht die resultierende Forschung nur indirekt in die Erarbeitung zukünftiger Alltagsroboter ein, da die Probleme und Gegebenheiten sehr speziell sind. Die Sportrobotik bietet allerdings viele Herausforderungen, die echte Anwendung in Unterhaltungs- oder Alltagsroboter finden. Die Interaktion mit- und in der realen Welt steht dabei im Vordergrund. Doggy ist ein ballspielender Roboter der im Grundgedanke die Felder Sport, Technologie und Interaktivität vereint und soll die angewandten Technologien auf verschiedenen Veranstaltungen demonstrieren.

2.3.1 Hardware

Der Roboter besteht aus einer Basis und einem Schläger die zusammen in einem genähten Hundekostüm stecken. Die gesamte Höhe beträgt etwa 221cm mit einem Schlägerlänge von 120,8cm und einer Basishöhe von 100,1cm. Die breitesten Stelle beträgt 50cm im Durchmesser. Der Schläger besteht aus einer Stange aus Kunststoff mit einer Styroporkugel als Kopf mit einem Durchmesser von 40cm und besitzt in der Bewegung 2 Freiheitsgerade. Zwei Kameras sind in einer Höhe von 83,6cm und einem 35 Grad-Winkel, vom Horizont aus, angebracht. Die Distanz zwischen den Kameramittelpunkten beträgt 44,7cm. Sie besitzen eine Brennweite von 4,8mm wodurch sich ein horizontales Sichtfeld von ungefähr 57 Grad ergibt. Der Schläger wird durch zwei Servomotoren in Roll- und Kippbewegung durch Riemen bewegt, die diesen mit jeweils knapp 10,5Nm Antriebsdrehmoment beschleunigen können. Effektiv wird der Schläger auf 70% Drehmoment mit 180 Grad/s angetrieben.

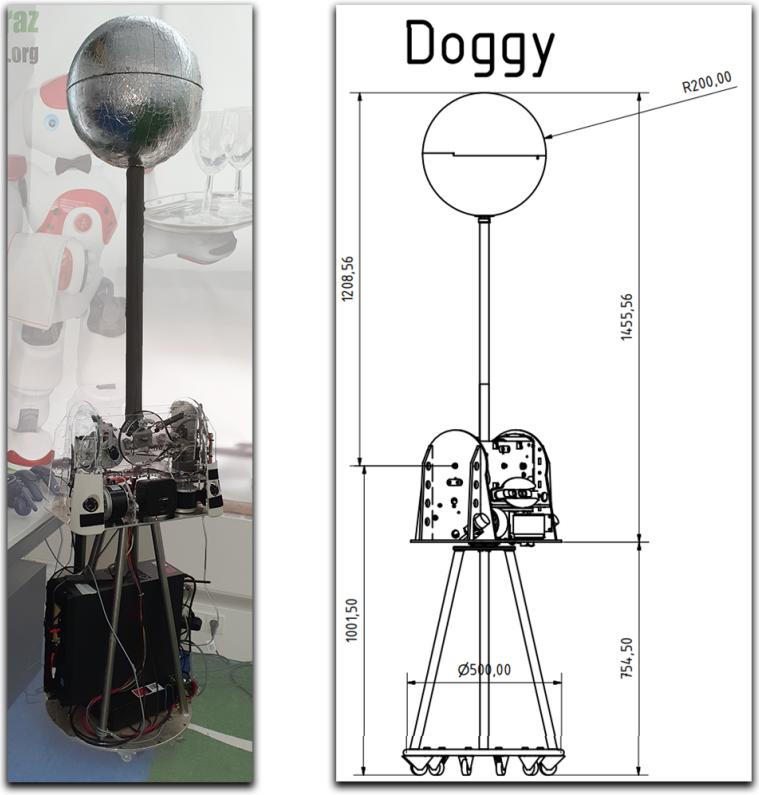

Abbildung 2.5 Realer Aufbau und Planungsskizze des Roboters Doggy der AG Multisensorische Interaktive Systeme.

Wichtige Faktoren beim Entwurf des Roboters waren eine hohe Sicherheit im Umgang mit dem Roboter zu gewährleisten und diesen zu einem relativ günstigen Preis anbieten zu können. Diese Einschränkungen führten dazu, dass leichte und leicht zu beschaffende Materialien verwendet wurden. Der bewegbare Teil des Roboters, also der Schläger, ist mit Schaumgummi ummantelt und der Kopf besteht aus Styropor. Diese Vorkehrungen wurden getroffen, um ein Verletzungsrisiko zu minimieren.

Der gesamte Roboteraufbau ist der Abbildung 2.5 zu entnehmen.

2.3.2 Software

Das gesamte System ist auf dem Robot Operating System Framework (ROS [12]) implementiert und läuft unter Ubuntu Linux (16.4). Jede Softwarekomponente ist ein eigener Prozess, der durch spezielle Nachrichten innerhalb ROS Daten verschicken und kommunizieren kann. Bilder der Kameras (768×576, 8 Bit Graustufen) werden mit 40 Hz aufgenommen, Kreise auf ihnen erkannt und einem Tracker übergeben. Dieser lässt den Kreiserkenner genauer auf vorhergesagten

Positionen suchen und verbessert seine Schätzung mit der neu erkannten Position. Die Positionskontrolle des Schlägers wird mit 100 Hz auf die Position gefahren, die der Tracker als Erste im Bewegungsradius des Roboters schätzt. Das gesamte Trackingsystem wurde 2013 von Birbach et al. [1] erarbeitet und benutzt ein physisches Bewegungsmodell zur Vorhersage der Flugbahn eines geworfenen Balles.

Die Erkennung der Bälle wird zuerst durch einen Kreiserkenner auf Bildern, dann durch einen Multiple Hypothesis Tracker (MHT) zum Feststellen der Position und Geschwindigkeit realisiert.

2.3.2.1 Kreiserkenner

Der Kreiserkenner verwendet eine Transformation zur Feststellung radialer Gradienten und untersucht gefundene Gradienten an ihren Konturen nach allen möglichen Kreispositionen und -größen. Die wahrscheinlichsten Hypothesen werden dem MHT übergeben. Der Vorteil dieser Methode ist, dass Belichtung der Szene und Farbe des Balles durch die verwendete Transformation keine Rolle spielen.

2.3.2.2 Multiple Hypothesis Tracker

Der MHT sucht für jedes verfolgte Ziel die Hypothese mit der größten Posterior Wahrscheinlichkeit. Die Hypothesen werden gebildet, indem nach jeder Messung jede Hypothese aus der vorherigen Iteration mit den Ergebnissen der aktuellsten Messung überprüft wird. Hypothesen werden für die Flugbahnen von Bällen gebildet, wobei vorab typische Informationen zum Beginn einer möglichen Flugbahn verwendet werden (Flugbahnen beginnen meistens vom unteren Rand des Bildes). Das letztendliche Errechnen von Hypothesen und die Verrechnung von Messungen werden von einem Unscented Kalman Filter [15] durchgeführt.

2.3.2.3 Bewegungskontrolle

Die Bewegung des Schlägers wird so geplant, dass der Mittelpunkt des anvisierten Balles den Mittelpunkt der Styroporkugel trifft. Erreicht wird dies durch die Berechnung der Winkel für die Servomotoren durch inverse Kinematik und anschließender Ansteuerung dieser Motorpositionen durch einen PD-Regler.

Kapitel 3

Verwandte Arbeiten

In der von Tsai et al. [14] erarbeiteten Mensch-Maschine Schnittstelle werden Hand-Gesten auf Kamerabildern erkannt. Um genügend Bilder für das Training eines Deep CNN zu verwenden, synthetisierten sie Kamerabilder und nutzten eine automatisierte Annotierung. Durch dieses kontrollierte Verfahren zur Datenbeschaffung können sie die Verteilung und Varianz der Daten bestimmen und auf den Kontext anpassen. Zur Erstellung der künstlichen Daten wurde eine 3D Hand modelliert und mit Gelenken zur kontrollierten Bewegung ausgestattet. Das System unterstützt 24 Gesten, die durch die Position der einzelnen Finger unterschieden werden. Dabei ist die Handinnenfläche der Kamera zugewandt und die Position eines einzelnen Fingers wird unterschieden in ausgestreckt, halb angewinkelt oder komplett angewinkelt. Durch das Training auf den synthetisierten Daten ergab sich bei der Evaluierung auf realen Kameradaten eine Accuracy von nur 37,5%. Durch das Hinzufügen von nur 0,09% realer Trainingsdaten, ist die Accuracy auf 77,08% gestiegen.

D'Orazio et al. [3] implementierten 2004 einen Algorithmus zur Ballerkennung mittels Houghtransformation für Kreise und einem neuronalen Klassifizierer. Dafür wurden Bildbereiche um erkannte Kreise ausgeschnitten und von dem binären, neuronalen Klassifizierer nach "Ball" oder "kein Ball" bewertet. Die ausgeschnittenen Bälle werden über eine 2D Wavelet Transformation nach ihren Frequenzen charakterisiert. Zudem erfolgte eine Implementierung zur Echtzeit-Erkennung auf graustufen Bildern der Größe 532×512. Sie kamen in verschiedenen Ansätzen der Vorverarbeitung auf Erkennungsraten von durchgehend über 92%.

S. Mitri, S. Frintrop, K. Pervölz, H. Surmann und A. Nuchter [9] wählten 2005 einen ähnlichen Ansatz zur Erkennung von Bällen im Kontext des RoboCup. Sie wählten einen biologischen Ansatz zur Perzeption der realen Welt und orientiert sich an dem Menschen. Zuerst wird die Aufmerksamkeit in Region des Sichtfeldes von objektspezifischen Merkmalen geweckt und erst dann eine Hypothese verifiziert. Auf 180 Bildern aus dem RoboCup Umfeld konnte der erarbeitete Klassifizierer alleine eine Accuracy von 81,1% mit einer Präzision von 47,7% erreichen. Zusammen mit dem entwickelten Aufmerksamkeitssystem wurde eine Accuracy von 78,3% und einer Präzision von knapp 86% erreicht.

Kapitel 4

Realisierung

4.1 Übersicht

Um das bestehende Ballerkennungssystem zu verbessern, ist es sinnvoll Fehlerkennungen von Bällen zu vermeiden. Hinter dem Ballerkenner im System verarbeitet ein Tracker die Mittelpunktpositionen von gefundenen Bällen und schätzt ihre Flugbahn. Dafür ist es wichtig, dass die Ballposition exakt erkannt wird. Der Tracker verbessert seine Schätzungen der zukünftigen Ballposition immer wieder mit den erkannten Positionen. Falsch erkannte Bälle und Ballmittelpunkte können diese Schätzung negativ beeinflussen.

Der aktuelle Ballerkenner ist auf der Methodik der *Hough-Transformation* für Kreise erarbeitet und erkennt lediglich Kreise und ihre Mittelpunkte im Eingabebild. Ein großes Problem dieser Methodik ist, dass eine große Menge an möglichen Kreisen mit verschiedensten Mittelpunkten gefunden wird. Bislang wird hier der wahrscheinlichste Kreis-Mittelpunkt ausgewählt, dieser kann allerdings von der Realität durch einfache Faktoren wie Kanten im Hintergrund oder Verzerrung des Balles stark beeinflusst sein.

Zur Verbesserung des bestehenden Erkennungssystems soll die Postion des Mittelpunkts mit einer Maske eingegrenzt werden. Ein CNN, welches als Eingabe das Kamerabild bekommt und als Ausgabe eine Bildmaske mit Wahrscheinlichkeiten der Mittelpunktpositionen pro Pixel liefert, die später über einen Schwellwert eingegrenzt und die Aggresivität der Maskierung angepasst werden können.

Anforderungen an das zu entwickelnde System sind:

Eingabe: (768, 576, 1)
Ausgabe: (768, 576, 1)
Zeit pro Prediction: <7ms

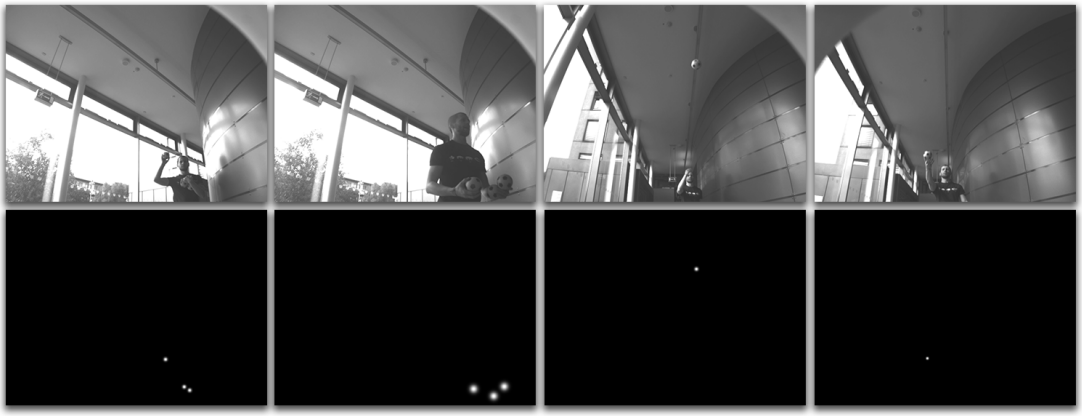

Abbildung 4.1 Auswahl realen Kamerabilder mit zugehöriger Annotation.

4.2 Eingesetzte Mittel

Das Netz und die Synthetisierung wurden mit Python 3 entwickelt, da ein schneller und einfacher Umgang mit Bibliotheken ermöglicht wird. Zudem ermöglicht die Software Jupyter Notebook ein interaktives Arbeiten an Code und verschiedenster Daten. Folgende Python-Bibliotheken werden in der Arbeit verwendet:

- keras
- tensorflow
- openCV
- numpy

4.3 Datensatz

In diesem Abschnitt werden die Unternehmungen zur Erstellung der relevanten Datensätze für diese Arbeit beschrieben.

4.3.1 Reale Daten

Zur Erstellung eines gemischten Datensatzes für die Erkennung von Ballmittelpunkten wurden zwei Sätze an Bildern am Roboter Doggy aufgenommen. Eins Auswahl davon ist auf Abbildung 4.1 mit zugehöriger Annotation zu sehen.

4.3.2 Synthetisierung von Bilddaten

In diesem Kapitel geht es um die spezifische Szenengenerierung durch bekannte Größen und Annahmen über das Zielsystem. Der Prozess der Bildgenerierung wurde im Rahmen dieser Arbeit erarbeitet und stetig erweitert. Um möglichst reale Bilder zu erzeugen, bietet es sich an, die Kamerabilder des Systems auf Merkmale zu untersuchen die nachgestellt werden müssen:

- Musterung des Balles
- Bewegungsunschärfe in Translation des Balles
- Bewegungsunschärfe in Rotation des Balles
- Belichtung des Balles
- Belichtung der Szene
- Bewegungspfad des Balles

4.3.2.1 Ballsynthese

Zur Synthetisierung der realen Bilddaten wurde im Rahmen dieser Arbeit eine Blender [2] 3D Szene erstellt, welche die Größen, Abstände und Winkel des realen Systems nachstellt. Ein modellierter Fussball wird über Keyframes entlang einem vorgegeben Pfad bewegt und zusätzlich um sich selbest rotiert. Der Bewegungspfad des Balles wird pro simulierten Wurf auf eine zufällige Position innerhalb des relevanten Sichtfeldes verschoben und gedreht. Eine Beleuchtung wird zusätzlich pro Wurf um den Ball verschoben, sodass eine im Wurf stetige, aber immer unterschiedliche, Beleuchtung gewährleistet wird. Der Vorteil bei der Benutzung von Keyframes ist, dass Bildserien erzeugt werden können, die einen gesamten Wurf simulieren und somit die Bilder eine Bewegungsunschärfe in Translation als auch Rotation des Balles aufweisen. Dies wird bei der Synthetisierung möglichst realer Daten benötigt, da diese gleiche Merkmale und Verzerrungen des Balles aufweisen. Ein Wurf wird durch 15 Keyframes simuliert, in denen der Ball zu Beginn steht und sich am Ende in der Nähe des Roboterkopfes befindet. Die Fussbälle werden dank der Blender Render Engine *Cycles* mit den, oben genannten, Unschärfen auf einen transparenten Hintergrund gerendert. Dies bringt zum einen den Vorteil mit sich, dass die Bilder später auf verschiedenste Hintergründe gelegt werden und somit wiederverwendet werden können, zum anderen ermöglicht es eine einfache Erzeugung der Annotation (4.3.3.2) mithilfe des gerenderten Alphakanal des Bildes. Die erarbeitete Blender 3D Szene ist als *ballThrowSimulationDoggy.blend* beiliegend, das Python Steuerungsskript für die Automatisierung liegt als *ballThrowControllDoggy.py* bei.

4.3.2.2 Szenengenerierung

Um eine komplette Szene mit den synthetisierten Bällen im Vordergrund zu erzeugen, fehlt dem generiertem Bild ein Hintergrund. Da der Roboter Doggy ein mobiles System ist, können wir

Abbildung 4.2 Auswahl der Hintergrundbilder für die Synthetisierung der Ballbilder.

nicht von einem stetigen Hintergrund ausgehen. Wir können trotzdem Annahmen durch das Einsatzgebiet und die Spezifikationen des Roboters über das Kamerabild treffen:

- Menschen werden dem System gegenüber stehen und Bälle werfen.
- Der Winkel, in dem die Kameras angebracht sind, lässt die Kameras viel Himmel und wenig Boden zeigen.
- Es werden graustufen-Kameras mit variabler Belichtungszeit verwendet.
- Es kann eine Extrembelichtung durch die Sonne auftreten, da das System auch im Freien eingesetzt wird.

Die Auswahl realer Hintergrundbilder erfolgte über eine Internetrecherche, um oben genannte Aspekte im resultierenden Datensatz möglichst abzudecken. Alle ausgewählten Bilder unterliegen keinerlei Lizenz, da sie von dem Archivbild-Anbieter *Pexels.com* stammen und dieser grundsätzlich nur lizenzfreie Bilder anbietet.

Da die ausgewählten Hintergrundbilder den ästhetischen Aspekten der Fotografie unterliegen und somit eher perfekte Voraussetzungen bieten, ist der Satz um Extrembeispiele erweitert. Diese sollen die Extrembelichtungen (Überbelichtung und Unterbelichtung) widerspiegeln, als auch eine Abstraktionsebene für das neu entwickelte System schaffen. Mit diesen Bildern wird versucht eine höhere Konzentration innerhalb des Bildes auf die Musterung und Form des Balles zu legen, als auf das Umfeld des Balles.

Abbildung 4.3 Erweiterung der Hintergrundbilder durch künstliche Szenen.

4.3.2.3 Komposition

Der letzte Schritt der Synthetisierung ist das Zusammensetzen von Vordergrund-Ball-Bild und Hintergrund-Szenen-Bild. Dafür werden diese beiden Bilder pixelweise verknüpft, wobei jeder Pixel des Szenenbild aus (R_b, G_b, B_b) und jeder Pixel des Ballbild aus (R_f, G_f, B_f, A_f) besteht. Die resultierenden Pixel bestehen aus (R_r, G_r, B_r) und setzen sich wie folgt zusammen:

$$\begin{pmatrix} R_r \\ G_r \\ B_r \end{pmatrix} = (1 - A_f) * \begin{pmatrix} R_b \\ G_b \\ B_b \end{pmatrix} + A_f * \begin{pmatrix} R_f \\ G_f \\ B_f \end{pmatrix} \tag{4.1}$$

Das erarbeitete Skript *generateTrainingImages.py* übernimmt die Komposition jedes Vordergrunds mit jedem Hintergrund.

4.3.3 Erzeugung der Annotationen

4.3.3.1 Annotation realer Daten

Um den Ballmittelpunkt auf den realen Bilddaten zu annotieren, wurde im Rahmen dieser Arbeit ein Labeltool in Python auf Basis der Bibliothek openCV implementiert. Dieses vereinfacht die Annotationsleistung durch eine einfache Bedienung per Maus und einer Live-Vorschau. Außerdem werden die Daten perfekt auf die Weiterverarbeitung vorbereitet. Das Labeltool ist der Arbeit als *labelMyImages.py* beiliegend. Ein Label besteht aus einer 2-Dimensionale Gaußglocke zusammen mit einem Kreis um die Gausglocke. Um eine perfekte Annotationsleistung zu bekommen, ist der Kreis des Labels deckend auf den zu annotierenden Ball zu legen. Es werden ein Ground-Truth- und ein Composed Bild für das Training des neuronalen Netzes erzeugt. Zudem ein Evaluationsbild für die Generierung einer Evaluations-CSV-Datei, um Mittelpunkt und Ball-

radius für die Evaluierung genau bestimmen zu können. Die Evaluations-CSV-Datei wird durch das erstellte Skript *generateEvaluationList.py* generiert.

4.3.3.2 Annotation synthetisierter Daten

Ziel der gesamten automatisierten Synthetisierung ist einerseits die Menge der Daten um ein vielfaches zu erhöhen, andererseits aber auch die Annotationen für die vielen Daten zu automatisieren. Angesetzt wird hier direkt bei den, von Blender generierten, Ballbildern mit transparentem Hintergrund. Der Vorteil hierbei ist, dass der Alphakanal der Bilder die exakte Position aller zum Ball gehörigen Pixel maskiert und somit zur Positionsbestimmung des Ballmittelpunktes genutzt werden kann. Dafür werden zuerst Konturen mit OpenCV auf der Alphamaske gesucht und deren Größe ermittelt. Der Ballmittelpunkt wird über den Schwerpunkt der Kontur ermittelt und mit einer 2-Dimensionalen Gaußglocke der Größe $\frac{Konturhoehe+Konturbreite}{4} \times \frac{Konturhoehe+Konturbreite}{4}$ im Zentrum markiert. Der Radius der Gaußglocke entspricht dem halben durchschnittlichen Radius des Balles. Mit der Gaußglocke ist auch die Ungenauigkeit des Mittelpunktes bei verzerrten Bällen modelliert. Durch die Annotierung der Bilder mit Bällen auf transparentem Hintergrund kann die Annotation für jedes zusammengesetzte Bild mit diesem Vordergrund verwendet werden, da sich die Ballposition durch die Komposition (4.3.2.3) nicht verändert. Hierfür ist das Skript *generateCenterGTImages.py* entwickelt worden, welches als Eingabe die Ballbilder auf transparentem Hintergrund bekommt, und als Ausgabe Ground-Truth Bilder liefert.

4.4 Ballerkennung mittels CNN

In diesem Abschnitt wird auf die Auswahl der CNN Architektur und das Training dieser eingegangen.

Durch die beiden Kameras werden im bestehenden System pro Sekunde 80 Frames (pro Kamera 40 Bilder/Sekunde) aufgenommen und dem bestehenden Ballerkenner geliefert. Um diese Bilddaten zu verarbeiten, ist die Architektur des Netzes auf wenige Schichten beschränkt. Um das Netz zusätzlich performant zu halten, ist es an dem MobileNetV2 orientiert. Dieses ist dafür optimiert, auf leistungsschwachen Geräten CNNs nutzbar zu machen. Um die Netzarchitektur dem MobileNetV2 nachzuempfinden, sind alle rechnenden Schichten durch Inverted-Residual-Blöcke implementiert. Grundlegend wird das Kamerabild mit einer viertel Auflösung in das Netz eingegeben und innerhalb des Netzes auf die sechzehntel Auflösung reduziert. Um möglichst viele Informationen über die Ballmittelpunkte zu erhalten, wird das Bild zurück auf die viertel Auflösung gerechnet. Dazu werden Informationen mit Querverbindungen aus früheren Aktivierungen mit den hochgerechneten Informationen aus dem Upscaling verkettet. Als Ausgabe hat das Netz ein Sigmoid-Aktivierungsbild mit gleichen Dimensionen zur Eingabe.

4.4.1 CNN-Modellfindung

Um ein genaues Modell zu finden, sind das Training und die Datenvorbereitung in diesem Kapitel beschrieben.

4.4.1.1 Augmentation

Aus der Konstruktion der Trainingsdaten kommt es oft vor, dass gleiche Vordergrundbilder (und somit Bälle) auf verschiedenen Hintergründen zu sehen sind. Insgesamt ist die Datenmenge ebenfalls relativ klein und somit für das Training untrainierter neuronaler Netze nicht optimal. Bei kleine Datenmengen für das Training tendieren die Lernverfahren dazu, den Trainingsdatensatz auswendig zu lernen (Overfitting) und erfüllen die eigentliche Aufgabe nicht. Besonders wichtig ist dies zu vermeiden bei Trainingsdaten, die nicht aus dem eigentlichen Anwendungsfeld stammen, da sich hier an vielen anderen Merkmalen wie besonderer Beleuchtung oder der Umgebung orientiert werden kann. Das Netz muss durch die Konstruktion der Trainingsbilder eine hohe Abstraktionsfähigkeit lernen. Perez et al., Taylor und Nitschke, Xu et al. [10, 13, 16] zeigen, dass die Augmentation von Trainingsdaten eine Lösung für dieses Problem bietet.

Bevor ein Trainingsdatum, hier konkret Trainingsbild, dem Lernalgorithmus zur Verfügung gestellt wird, wird zufällig eine Menge an Transformationen auf dieses angewandt. Somit kann sich bei jeder Epoche die Beleuchtung, das Rauschen und die Ausrichtung dieses Datums ändern.

- **Spiegelung**
  Eine horizontale oder vertikale Spiegelung des Bildes verändert weder Kontext noch visuelle Eigenschaft des Bildes und vergrößert die Trainingsmenge und ihre Varität. Die selbe Transformation muss hier auch auf die Annotation angewandt werden, da sich der Ball innerhalb der Bilddimension bewegt.
- **Unschärfe des Bildes**
  Durch die Anwendung eines Gauß-Filters kann zusätzlich eine Unschärfe des gesamten Bildes erreicht werden, wodurch scharfe Kanten weichgezeichnet werden. Die Annotation kann hier unverändert bleiben.
- **Helligkeitsänderungen**
  Durch verschiedene Beleuchtungen auch innerhalb einer Wurfsequenz in der Realität, ausgelöst durch Verdeckung der Sonne durch den Ball oder die Umgebung, kann die Helligkeit der Kamerabilder variieren. Zudem kann durch Änderung des ISO-Wertes der Kamera eine Extrembelichtung auftauchen. Durch eine Addition auf alle Pixel des Bildes kann eine Gesamtänderung hervorgerufen werden, durch eine Multiplikation können explizit helle oder dunkle Bereiche verstärkt werden. Da keine Positionsänderung des Balles innerhalb der Bilddimension auftritt, kann die Annotation unverändert bleiben.

Modell	Parameter	Loss	MAE
BallDet-Real	7853	$8,1057 \times 10^{-4}$	$5,0715 \times 10^{-4}$
BallDet-Mid	4527	$8,7934 \times 10^{-4}$	$5,4690 \times 10^{-4}$
BallDet-Small	3385	$7,9047 \times 10^{-4}$	$4,4646 \times 10^{-4}$

Tabelle 4.1 Vergleich der Performanz von 3 erarbeiteten Netzen auf den Evaluierungsdaten.

4.4.1.2 Training

Zur Bestimmung des besten Modells für die Aufgabe wurde ein iterativer Ansatz gewählt. Es wurden verschiedene Versuche durchgeführt, bei denen das Modell angepasst wurde um Parameter wie Kanalzahl oder Blockwiederholung zu finden. Zum Training der Modelle wurde eine Batch-Size von 4 gewählt und jeweils für 25 Epochen trainiert. Um das beste Modell zu finden wurde auf einem Datensatz, bestehend aus 200 realen Kamerabildern und 2800 synthetisierten Bildern, trainiert. Für die Optimierung der Parameter des Netzes beim Training wird der Adam-Optimierer benutzt.

An den Daten aus Tabelle 4.1 ist gut zu erkennen, dass das kleinste Netz *BallDet-Small* trotz weniger Parameter nach 25 Trainingsepochen eine bessere Leistung auf dem Evaluationsdatensatz liefert.

4.4.1.3 Auswahl

Aus Performanzgründen, welche während der Integration auffielen, zusammen mit dem Ergebnis des Trainings der Netze, wurde sich innerhalb dieser Arbeit für die standardmäßige Benutzung des *BallDet-Small* entschieden. Eine genaue Auflistung der Schichten und ihren Kanalparametern ist in Abbildung 4.4 zu sehen.

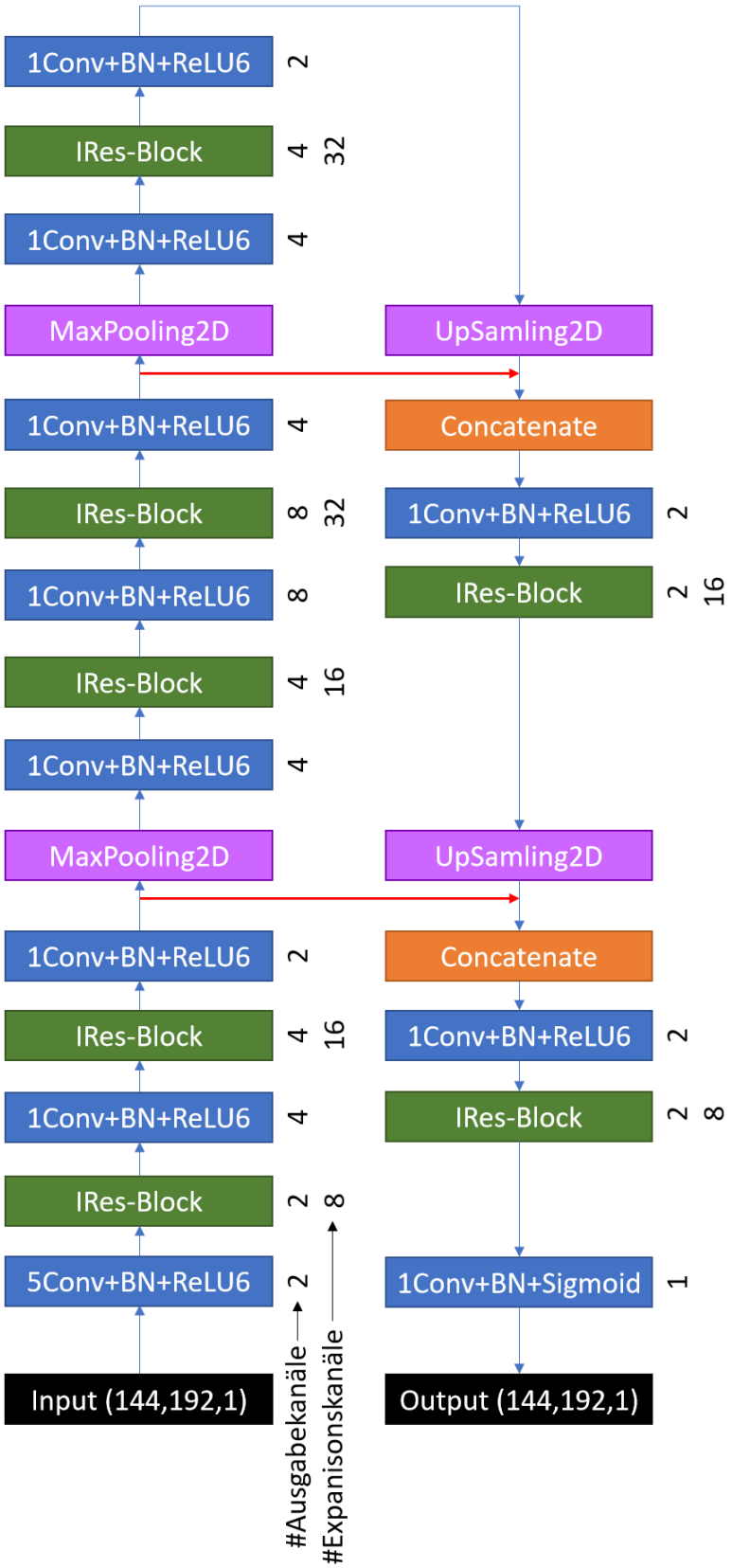

Abbildung 4.4 Das erarbeitete Netz *BallDet-Small* mit allen Schichten und Kanalparametern. Es ist einen Abwandlung des *BallDet-Real* Netz mit gleicher Schichtenanzahl aber weniger Kanälen.

Kapitel 5

Integration

Um das erarbeitete Modell in das bestehende System (Abschnitt 2.3) zu integrieren sind folgende Schritte unternommen worden.

5.1 Einordnung im System

Da das bestehende System in ROS Knoten entwickelt wurde, besitzt es eine hohe Modularität. In Abbildung 5.1 sind in blauer Farbe die ROS Knoten zu sehen, in oranger Farbe ausgelagerte Bibliotheken. Um den Kreiserkenner (CircleDetector) zu verbessern, wird dieser Knoten modifiziert. Das restliche System bleibt unverändert. Im Repository der Doggy Software ist ein zusätzlicher ROS Knoten mit dem Namen *CircleDetectorCNN* erstellt.

5.2 Erweiterung des Systems

Zur Integration eines neuronalen Netzes in den C++ Code des Kreiserkenners, wird die Bibliothek *TensorFlow Lite* [8] verwendet. Diese ist speziell auf die ressourcensparende Nutzung neuronaler Netze in rechenschwachen Systemen ausgelegt. Eine TensorFlow Lite Inference beschreibt den Prozess der Ausführung eines TensorFlow Lite Modells. Dafür muss das Modell durch einen Interpreter laufen, der eine statischen Graphreihenfolge und einen besonderen Speicherallokator benutzt. So können minimale Anforderungen an das System im Zuge minimaler Last, Initialisierungs- und Ausführungslatenzzeit gestellt werden. Das trainierte Modell wurde in ein TensorFlow Lite Modell konvertiert und wird bei der Initialisierung des ROS Knoten mit einem TensorFlow Lite Interpreter geladen.

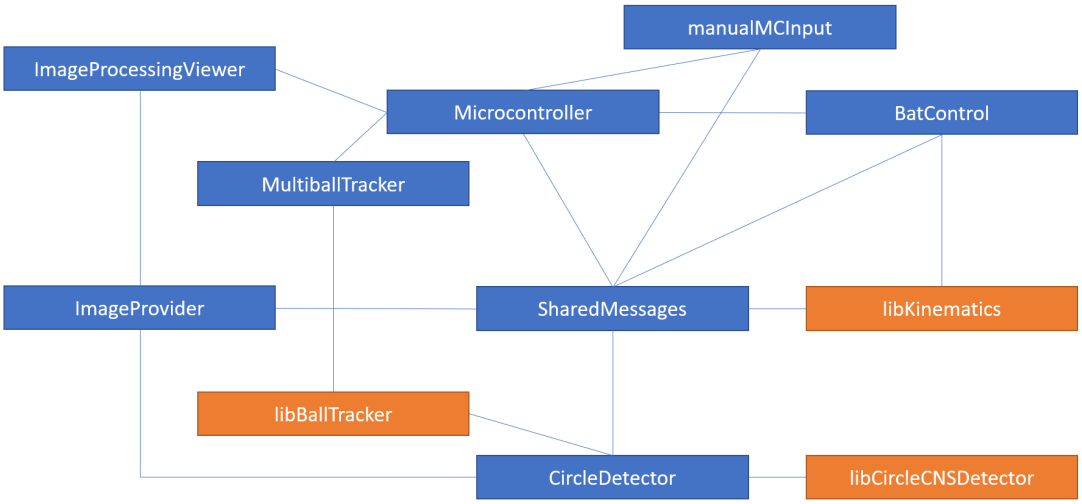

Abbildung 5.1 Übersicht über die bestehenden ROS Knoten (blau), verwendete Bibliotheken (orange) und Kommunikationskanäle untereinander.

5.2.1 Maskenerstellung

Die Kamerabilder werden beim Aufruf des Kreiserkenners für das Netz verkleinert, in den Interpreter geladen und dieser Aufgerufen. Die Ausgabe des Modells beinhaltet die letzte Sigmoid-Aktivierung und somit Werte von 0 bis 1. Für die Weiterverarbeitung muss das Ausgabebild wieder vergrößert werden. Der Kreiserkenner besitzt eine bestehende Funktion zur Nutzung von Bildmasken zur Bereichseinschränkung der Mittelpunkterkennung. Die erwarteten Bildmasken müssen die selbe Größe wie die Kamerabilder besitzen. Ein schwarzer Farbwert (Wert 0) in einem 1 Kanal 8 Bit Bild sagt aus, dass an dieser Position kein Mittelpunkt zu Suchen ist, ein weißer Farbwert (Wert 255) maskiert eine valide Position für einen Kreismittelpunkt. Um aus der Ausgabe des Netzes eine valide Maske für diese Funktion zu Erstellen, wird ein Schwellwert definiert. Alle Pixelwerte die größer oder gleich dem Schwellwert sind, werden zu einer validen Position und somit auf Wert 255 gesetzt. Alle Werte die unter den Schwellwert fallen werden in der Bildmaske auf 0 gesetzt. Der Schwellwert ist variierbar und ergänzt die Funktion der 2D Gaußglocke, die bei der Erstellung der Annotationen verwendet wurde. Durch den Schwellwert ist die Größe der Mittelpunktregion einstellbar, da durch Ihn die Größe des Bereichs um die Mitte einer Gaußglocke selektiert werden kann.

Kapitel 6

Evaluation

Für die Evaluation des erarbeiteten Systems wird der bestehende Kreiserkenner gegen den verbesserten Kreiserkenner mit CNN generierter Maske auf den selben Kamerabildern laufen gelassen. Gefundene Bälle werden mit Mittelpunktposition (X- und Y-Koordinate) und Radius zusammen mit dem Dateinamen des Kamerabildes in einer CSV-Datei gespeichert. Ein entwickeltes Evaluationsskript lädt die Informationen dieser CSV-Datei und der Annotationsdatei (Siehe Abschnitt 4.3.3.1). Um einen annotierten Ball als True-Positive markieren zu können, wird eine Formel (6.1) zur Fehlerberechnung angewandt. In diese fließen die erkannten Merkmale Position (x, y) und Radius (r) ein. In dieser Evaluation ist für eine True-Positiv Erkennung ein maximaler Fehler von 6 definiert.

$$error = \sqrt{(x_{found} - x_{truth})^2 + (y_{found} - y_{truth})^2} + |r_{found} - r_{truth}| \tag{6.1}$$

6.1 Maße

Als Qualitätsmaße für die Erkennung von den Bällen werden die Maße *Precision* (Gleichung 6.3) und *Recall* (Gleichung 6.4) herangezogen. Die Precision beantwortet die Frage, wie viele korrekte Erkennungen das System liefert. Der Recall zeigt, wie viele geworfene Bälle erkannt werden. Das Maß *Accuracy* findet in dieser Evaluation keine Anwendung, da dieses True-Negatives in die Berechnung mit einbezieht (siehe Gleichung 6.2). True-Negatives sind im Falle des Kreiserkenners alle möglichen Pixel des Bildes und Radien an denen sich kein Ball befindet, und der Kreiserkenner auch keinen erkennt. Aus der Konstruktion der zu erkennenden Kreise heraus ist dieses Maß nicht definiert, da es unendlich Radien an jeder Pixelposition des Bildes gibt. Um trotzdem ein Maß für die Accuracy zu erhalten, stützen wir uns dafür auf den F_1-*Score* (Gleichung 6.5), der sich aus dem harmonische Mittel von *Precision* und *Recall* ergibt.
In folgenden Formel sind definiert:
TP = True-Positive, TN = True-Negative, FP = False-Positive, FN = False-Negative

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{6.2}$$

$$Precision = \frac{TP}{TP + FP} \tag{6.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{6.4}$$

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{6.5}$$

6.2 Ergebnis

Zur Evaluation wurde ein weiterer Datensatz aufgenommen und annotiert. Dieser besteht aus 1728 Bildern mit insgesamt 686 annotierten Bällen. Auf den Bildern sind maximal 2 Bälle gleichzeitig zu sehen. Um die Maße zu berechnen wurden die beiden Kreiserkenner zwei Mal auf dem Datensatz laufen gelassen. Beim ersten Durchlauf durften die Systeme bis zu 10 Bälle auf einem Bild erkennen. Diese Konfiguration ist für den Betrieb auf Veranstaltungen mit vielen Bällen gedacht. Der zweite Durchlauf erlaubte die Erkennung von bis zu 2 Bällen auf einem Bild, um die False-Positive Rate der Erkennung einzuschränken. Dies ist sinnvoll, da der bisherige Kreiserkenner immer die 10 wahrscheinlichsten Bälle ausgegeben hat, unabhängig davon wie wahrscheinlich diese sind. Die Einschränkung wird in der Evaluierung *maxBälle* genannt.

In der Tabelle 6.1 ist zu sehen, dass bei der Einschränkung auf maximal 2 erkannten Bällen pro Bild das verbesserte System eine 4x höhere Precision als das bestehende System erreicht. Die Verbesserung bewirkt außerdem einen 3x höheren Recall, wodurch sich ein 3,8x besserer F_1-Score ergibt. Bleibt die Einschränkung bei 10 Bällen pro Bild, liefert das verbesserte System eine über 3x höhere Precision. Der Recall ist beim bestehenden System um 1,2x höher. Trotzdem ist der F_1-Score beim verbesserten System knapp 3x höher.

In Tabelle 6.2 sind die absoluten Ergebnisse der beiden Evaluationsdurchläufe aufgelistet.

Mit einer Messung der Laufzeiten pro Bild ist eine durchschnittliche Erkennungslaufzeit von 70ms festgestellt.

Erkenner	maxBälle	Precision	Recall	F_1
CD	10	0.010915	0.269679	0.020982
CD + CNN	10	0.034666	0.218658	0.059844
CD	2	0.010706	0.053936	0.017865
CD + CNN	2	0.043170	0.177842	0.069476

Tabelle 6.1 Ergbnisse der evaluierten Maße, bester Wert pro Spalte unterstrichen. *maxBälle* beschreibt die maximale Anzahl der Bälle, die der Kreiserkenner pro Bild finden darf.

Erkenner	maxBälle	True-Positive	False-Positive	False-Negative
CD	10	185	16763	501
CD + CNN	10	150	4177	536
CD	2	37	3419	649
CD + CNN	2	122	2704	564

Tabelle 6.2 Statistik über TP, FP und FN bei den Evaluierungen. *maxBälle* beschreibt die maximale Anzahl der Bälle, die der Kreiserkenner pro Bild finden darf.

Kapitel 7

Fazit und Ausblick

Das Ziel dieser Arbeit war es, den bestehenden Kreiserkenner des ballspielenden Roboters Doggy der AG Multisensorische Interaktive Systeme in der Erkennungsleistung zu verbessern. Das entwickelte System stützt sich zur Verbesserung auf die Maskierung von Bällen, das durch die Erkennung dieser durch ein neuronales Netzes erreicht wird. Die Ergebnisse der Evaluierungen zeigen, dass diese Erkennungsleistung einerseits nicht perfekt ist, andererseits die Erkennung stark verlangsamt. Die Precision des Erkenners wurde zwar um das 3,5-fache verbessert, der Recall ist dabei aber um das 0,8-fache gesunken. Die gewünschte Verbesserung des bestehenden Systems würde sich durch eine Verbesserung der Precision ohne eine Verschlechterung des Recalls äußern. Auf dem Evaluierungssystem, welches aus Gründen der vorherrschenden COVID-19 Pandemie während der Entstehung dieser Arbeit nicht das Zielsystem ist, generiert der verbesserte Ballerkenner ein Ergebnis pro Bild ungefähr 10x langsamer als die alte Ballerkennung. Die realistische Synthetisierung der Bilddaten hat, nach den Ergebnissen dieser Arbeit, eine stark positive Auswirkung auf die Erkennungsleistung gehabt. Testweise trainierte Netze, die nur auf den realen Datensätzen gelernt haben, zeigten eine nicht annähernd vergleichbare und nutzbare Leistung. Deutlich zu sehen an den Ergebnissen in 6.2 ist, dass die meisten Bälle für den Kreiserkenner nicht die wahrscheinlichsten Kreise sind. Durch die Einschränkung auf 2 Ballerkennungen pro Bild werden von den Systemen nur die beiden wahrscheinlichsten Kreise erkannt, worunter die meisten Bälle anscheinend nicht fallen. Ob das entwickelte System in der Form im Roboter nutzbar ist, wird sich in Versuchen mit dem integrierten Netz zeigen. Durch die große Abweichung zwischen Evaluierungssystem und Zielsystem ist zwar mit schnelleren Ausführungszeiten zu rechnen, jedoch wird die Erkennungsleitung pro Sekunde nicht in den angestrebten Bereich fallen. Eine denkbare Verbesserung der Genauigkeit des Systems, wäre ein tieferes Netzmodell mit mehr Auflösungsebenen, da damit Verwirrungen des Netzes an trivialen Stellen des Bildes ausgeschlossen werden können. Dies würde allerdings die Geschwindigkeitsproblematik noch mehr in den Vordergrund stellen. Die Synthetisierung der Würfe auf Kamerabilder in diesem System hat dennoch funktioniert und der erarbeitete Prozess kann für die virtuelle Evaluierung des Robotersystems genutzt werden.

Anhang

Datei	Beschreibung
thesis.zip	Diese Arbeit als PDF Datei und verwendete Bilder
script.zip	Skripte zur Synthetisierung und Automatisierung
model.zip	Erarbeitete Keras- und TensorFlow Lite Modelle
ballThrowSimulationDoggy.blend	Blender 3D Szene

Literatur

[1] O. Birbach und U. Frese. »A precise tracking algorithm based on raw detector responses and a physical motion model«. In: *2013 IEEE International Conference on Robotics and Automation*. Mai 2013, S. 4761–4766. DOI: *10.1109/ICRA.2013.6631255*.

[2] Blender Online Community. *Blender - a 3D modelling and rendering package*. Stichting Blender Foundation, Amsterdam: Blender Foundation, 2018. URL: *http://www.blender.org*.

[3] T. D'Orazio, C. Guaragnella, M. Leo und A. Distante. »A new algorithm for ball recognition using circle Hough transform and neural classifier«. In: *Pattern Recognition* 37.3 (2004), S. 393–408. ISSN: 0031-3203. DOI: *https://doi.org/10.1016/S0031-3203(03)00228-0*. URL: *http://www.sciencedirect.com/science/article/pii/S0031320303002280*.

[4] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto und Hartwig Adam. »MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications«. In: *CoRR* abs/1704.04861 (2017). arXiv: *1704.04861*. URL: *http://arxiv.org/abs/1704.04861*.

[5] Tim Laue, Oliver Birbach, Tobias Hammer und Udo Frese. »An Entertainment Robot for Playing Interactive Ball Games«. In: Jan. 2014, S. 171–182. ISBN: 978-3-662-44467-2. DOI: *10.1007/978-3-662-44468-9_16*.

[6] Y. LeCun, L. Bottou, Y. Bengio und P. Haffner. »Gradient-Based Learning Applied to Document Recognition«. In: (1998).

[7] H. Li, Y. Liu, X. Zhang, Z. An, J. Wang, Y. Chen und J. Tong. »Do we really need more training data for object localization«. In: *2017 IEEE International Conference on Image Processing (ICIP)*. 2017, S. 775–779.

[8] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu und

Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: *https://www.tensorflow.org/*.

[9] S. Mitri, S. Frintrop, K. Pervolz, H. Surmann und A. Nuchter. »Robust Object Detection at Regions of Interest with an Application in Ball Recognition«. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, S. 125–130.

[10] Luis Perez und Jason Wang. »The Effectiveness of Data Augmentation in Image Classification using Deep Learning«. In: *CoRR* abs/1712.04621 (2017). arXiv: *1712.04621*. URL: *http://arxiv.org/abs/1712.04621*.

[11] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov und Liang-Chieh Chen. »Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation«. In: *CoRR* abs/1801.04381 (2018). arXiv: *1801.04381*. URL: *http://arxiv.org/abs/1801.04381*.

[12] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. Version ROS Melodic Morenia. 23. Mai 2018. URL: *https://www.ros.org*.

[13] Luke Taylor und Geoff Nitschke. »Improving Deep Learning using Generic Data Augmentation«. In: *CoRR* abs/1708.06020 (2017). arXiv: *1708.06020*. URL: *http://arxiv.org/abs/1708.06020*.

[14] C. Tsai, Y. Tsai, S. Hsu und Y. Wu. »Synthetic Training of Deep CNN for 3D Hand Gesture Identification«. In: *2017 International Conference on Control, Artificial Intelligence, Robotics Optimization (ICCAIRO)*. 2017, S. 165–170.

[15] E. A. Wan und R. Van Der Merwe. »The unscented Kalman filter for nonlinear estimation«. In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*. 2000, S. 153–158.

[16] Yan Xu, Ran Jia, Lili Mou, Ge Li, Yunchuan Chen, Yangyang Lu und Zhi Jin. »Improved Relation Classification by Deep Recurrent Neural Networks with Data Augmentation«. In: *CoRR* abs/1601.03651 (2016). arXiv: *1601.03651*. URL: *http://arxiv.org/abs/1601.03651*.

[17] Xiangxin Zhu, Carl Vondrick, Charless C. Fowlkes und Deva Ramanan. »Do We Need More Training Data?« In: *CoRR* abs/1503.01508 (2015). arXiv: *1503.01508*. URL: *http://arxiv.org/abs/1503.01508*.