**Staatliche Fachoberschule München-West**

Seminararbeit

**Thema:**

*Vergleich der Umsetzbarkeit und Performance von Objekterkennung auf Microcontrollern: Eine Untersuchung der Pixy2-Kamera mit Arduino und der Implementierung eines TensorFlow Lite Modells auf einem Raspberry Pi.*

Vorgelegt von: *Paul Nöhles*

Vorgelegt am: *14.01.2025*

Fach: *Technologie, Informatik, Physik*

Rahmenthema: *Arduino im Alltag – innovative Anwendungsmöglichkeiten*

Seminarleitung: *Herr Ciccone; Frau Meurer*

**Inhalt**

[1 Einleitung 1](#_Toc187740129)

[2 Pixy2-Kamera mit Arduino 3](#_Toc187740130)

[2.1 Anschaffung 3](#_Toc187740131)

[2.2 Einrichten 3](#_Toc187740132)

[2.3 Pixy2 mit dem Computer verbinden 3](#_Toc187740133)

[2.4 Pixy mit Arduino verwenden 4](#_Toc187740134)

[2.4.1 Arduino Test Code 5](#_Toc187740135)

[2.5 Wie Object-Detektion mit dem Pixy2 funktioniert 6](#_Toc187740136)

[2.6 Problem und Performance der Pixy2-Kamera 6](#_Toc187740137)

[3 Tflite Model auf Raspberry Pi 7](#_Toc187740138)

[3.1 TFLite auf Raspberry Pi installieren 7](#_Toc187740139)

[3.2 TFLite Modelle auf Raspberry Pi verwenden 7](#_Toc187740140)

[3.3 Eigenes Model im TFLite Format für den Raspberry Pi trainieren 8](#_Toc187740141)

[3.3.1 Trainings Daten erstellen 9](#_Toc187740142)

[3.3.2 Versuch Model in Google Colab zu trainieren 10](#_Toc187740143)

[3.3.3 TensorFlow lokal aufsetzen 10](#_Toc187740144)

[3.3.4 Parameter meines Modells 11](#_Toc187740145)

[3.3.5 Training des Modells 12](#_Toc187740146)

[3.3.6 Model zu Tflite Format konvertieren 13](#_Toc187740147)

[3.4 Model mit Quantisierung optimieren 14](#_Toc187740148)

[3.5 Performance des selbst trainierten KI-Modells 14](#_Toc187740149)

[3.5.1 Der mAP meines Modells 14](#_Toc187740150)

[3.5.2 Die Geschwindigkeit meines Modells 15](#_Toc187740151)

[3.6 Verbesserung des Modells durch neu Training mit mehr Bildern 15](#_Toc187740152)

[4 Fazit 18](#_Toc187740153)

[5 Literaturverzeichnis 19](#_Toc187740154)

[6 Abbildungsverzeichnis 20](#_Toc187740155)

[7 KI-Verzeichnis 21](#_Toc187740156)

[8 Anhang 22](#_Toc187740157)

[9 Eidesstattliche Erklärung 23](#_Toc187740158)

# Einleitung

Künstliche Intelligenz hat in den letzten Jahren rapide Fortschritte gemacht und dadurch stark an Bedeutung gewonnen. Weshalb diese Technologie ein immer prominenterer Teil unseres Lebens wird und es wirkt nicht als ob diese Entwicklung nachlässt. Da KI es möglich macht viele Prozesse zu automatisieren bei denen das bisher nicht oder nur schwer möglich war.

Eine der interessantesten Anwendung von KI ist die Möglichkeit Maschinen mit KI das sehen bei zu bringen, eine wichtige Methode dafür ist Object-Detection auf Deutsch Objekterkennung. Objekterkennung ist eine Computer-Vision-Technik die Objekte in einem Video oder Bild lokalisiert und erkennt. Object-Detection-Algorithmen benutzen normalerweise maschinelles Lernen um nutzbare Ergebnisse zu produzieren. Wenn Menschen Bilder oder Videos betrachten, können wir nach einem kurzen Moment interessante Objekte lokalisieren. Das Ziel von Objekterkennung ist diese Intelligenz mit einem Computer zu replizieren (MathWorks, kein Datum). Diese Technologie wird in Zukunft zum Beispiel bei selbst fahrenden Autos noch eine Wichtige Rolle spielen, da diese Hindernisse im Straßenverkehr erkennen und deren Position bestimmen müssen um ihnen ausweichen zu können.

Aber auch schon jetzt lassen sich mit Objekterkennung viele interessante Anwendungen umsetzen. Zum Beispiel kann man selbst kleine DIY-Projekte (  
Selbstbastelprojekt) mit Objekterkennung mit Microkontrollern wie dem Raspberry Pi und dem Arduino verwirklichen. Dies ist ein besonders interessanter Anwendungsbereich da Microkontroller nicht wirklich das ideale technische Gerät für KI sind. Das Problem mit KI auf Microkontrollern ist, dass sie durch ihre stark begrenzte Hardwareleistung Probleme damit haben die Leistung intensiven KI-Anwendungen auszuführen. Will man auf ihnen Objekterkennungs-Modelle in Echtzeit ausführen kommt man dann schnell an Performancegrenzen. Zwar werden Prozessoren immer Leistung fähiger, kleiner und billiger, aber die neueren KI-Modell werden auch immer Leistungs hungriger. Weil diese Geräte Preisewert sein müssen werden sie nie die Fähigkeit haben die momentan besten KI-Modelle auszuführen.

Einen Weg diese Beschränkung zu Umgehen ist es die Berechnungen in der Cloud durchzuführen anstatt auf dem Gerät selbst. Diese Lösung ist aber nicht perfekt da dann der Mikrokontroller ständig mit dem Internet verbunden sein muss, was nicht bei allen Anwendungen immer möglich ist. Außerdem kommen bei Cloud Anwendungen immer Latenzen dazu die vor allem bei Anwendungen wie Objekterkennung, die in Echtzeit ausgeführt werden, deren Performance nach unten ziehen kann. Zusätzlich sind alle Daten die in der Cloud gespeichert werden einem deutlich höheren Risiko ausgesetzt kompromittiert zu werden als Daten die auf Endgeräten gespeichert sind. Was vor allem bei Anwendungen für den Privaten Haushalt eine wichtige Rolle spielt.

Deshalb wird daran geforscht KI-Modelle weniger Leistung intensiv zu machen und sie zu optimieren, damit sie auch auf weniger Leistungs starken Endhardware wie Microcontroller Laufen können. Diese Forschung ist aber nicht nur für Microkontroller wichtig, da wir eines Tages KI in vielen unser Alltags Geräte verbauen wollen. Zum Beispiel könnte es ihn Zukunft Kühlschränke geben die selbst mithilfe einer Kamera und Objekterkennung feststellen können, was in ihnen steht oder statt Haustürschlüsseln smarte Türen die über Gesichtserkennung verfügen. Diese Geräte müssen aber Preislich erschwinglich bleiben, um weite Verbreitung zu finden und werden deshalb auch nur über sehr schwache Hardware verfügen.

Deshalb habe ich mir in dieser Arbeit angeschaut wie gut man Objekterkennung auf Microcontrollern selbst umsetzen kann und die Performance von ihnen. An zwei konkreten Beispielen einmal dem Pixy2 smart Vision Sensor mit einem Arduino und einem TFLite Modellen auf einem Raspberry Pi mit Kamera.

# Pixy2-Kamera mit Arduino

Eine der Varianten die ich mir angeschaut habe um Objekt Detektion in DIY-Projekten umzusetzen war die Pixy2-Kamera in Kombination mit einem Arduino. Pixy ist ein smart Vision Sensor der selbst Objekterkennung von einfarbigen Objekten durchführen kann. Der jetzt mit dem Pixy2 in seiner zweiten Generation ist. In Kombination mit einem Arduino lassen sich mit ihm viele kleinere Projekte mit Objekterkennung verwirklichen. Da die Berechnungen der Objekterkennung nicht auf dem Arduino stattfinden ist es egal welchen Arduino man nimmt. Ich habe Pixy2 mit dem Arduino Nano und dem Arduino uno R4 WIFI getestet.

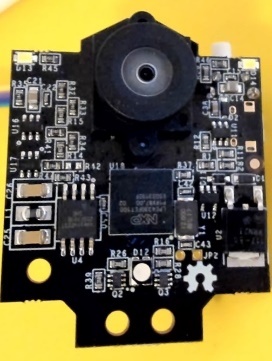


Abbildung 1 Bild vom Pixy2 eigene Aufnahme

## Anschaffung

Der Pixy smart Vision Sensor ist im Vergleich zu anderen Sensoren recht teuer da er einen eigen kleinen Prozessor besitzt. Er braucht diesen um das Objekterkennungs-Programm auszuführen. An den Arduino übermittelt er nur was er erkennt. Ich habe den Pixy2 Vision Sensor für 80,10 € gekauft, womit er deutlich teurer als der Arduino selbst war. Beim Kaufen sollte man beachten, dass es zwei Version gibt. Eine für Microkontroller und eine für den Lego Roboter Lego Mindstorm EV3.

## Einrichten

Als ich meinen Pixy2 erhalten habe musste ich feststellen, dass ich ausversehen die Lego Version und nicht die Arduino Version erworben hatte. Da ich die Lego-version gekauft hatte musste ich die Firmware meines Pixy2 mit der Firmware der nicht Lego-version überschreiben. Dies war möglich da die Modelle Baugleich sind. Dafür konnte ich einfach die Firmware von der Webseite des Herstellers herunterladen und dann auf das Pixy aufspielen. Indem ich das Pixy über ein USB-Kabel an meinen Computer anschließe und während dessen denn Knopf am Pixy gedrückt hielt.

## Pixy2 mit dem Computer verbinden

Um den Pixy2 Sensor an einem Computer zu benutzen kann man es mit einem USB A zu Micro USB-Kabel mit dem PC verbinden und muss die Software „PixyMon v2“ vom Hersteller runterladen und kann dann, wenn der Pixy-Sensor verbunden ist in PixyMon sehen was die Kamera des Pixys sieht, sowie die Objekte die es erkennt. Das funktioniert auch wenn die Pixy2-Kamera mit dem Arduino verbunden ist. Was recht nützlich beim Entwickeln mit dem Arduino ist, um zu sehen ob der Arduino auch die richtigen Werte übermittelt bekommt.

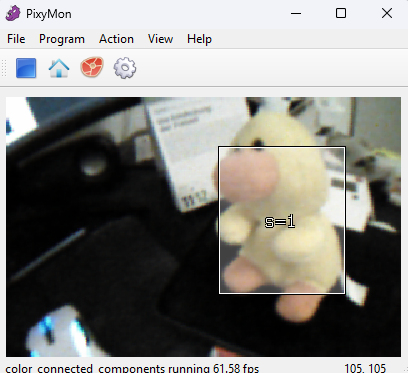


Abbildung 2 Screenshot von PixyMon eigene Aufnahme

In PixyMon kann man auch die einfarbigen Objekte einspeichern die der Sensor erkennen soll. Objekte kann man über mehrere Wege einspeichern. Entweder man zieht ihn PixyMon ein Viereck um das Objekt das man einspeichern will oder man hält das Objekt vor das Pixy und drückt den Knopf auf dem Pixy und lässt ihn los, wenn die Lampe am Pixy in der Farbe des Objekts Leuchtet. Die Methode mit dem Knopf neue Objekte hinzuzufügen funktioniert immer auch wenn die Pixy-Kamer nicht mit dem Computer verbunden ist. Auch könne wir in PixyMon die Toleranz der Farbeerkennung der einzelnen Objekte einstellen.

## Pixy mit Arduino verwenden

Da ich die Lego Version gekauft hatte, hatte ich nicht das Kabel um das Pixy mit dem Arduino zu verbinden, das bei der normalen Version dabei ist, sondern ein Kabel um es mit dem Lego Mindstorm EV3 zu verbinden. Weswegen ich mir aus weiblich weiblich Jumper Kabeln selbst ein Kabel gebastelt habe. Indem ich mir angeguckt habe welche Pins des Pixys zu den Pins des ICSP Anschluss des Arduinos über den es mit dem Arduino verbunden werden muss zusammengehören. Mit diesem selbst gebauten Kabel konnte ich dann die Pixy-Kamera mit dem Arduino verbinden.

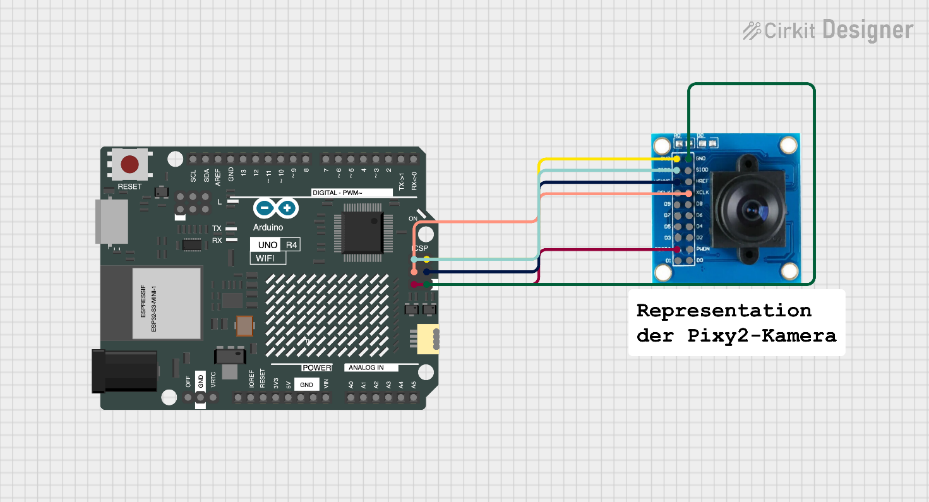


Abbildung 3 Schaltplan Arduin mit Pixy eigene Darstellung

Um mit dem Pixy dann auch kommunizieren zu können musste ich auch die Arduino Pixy2 Library in der Arduino IDE installieren. Die einem dann die Möglichkeit gibt die Signale die der Pixy Sensor schickt zu verarbeiten. Die Library konnte ich als ZIP Datei von der Webseite des Herstellers herunterladen und dann der Arduino IDE hinzufügen.

### Arduino Test Code

Ich habe dann getestet ob der Arduino mit dem Pixy auch funktioniert mithilfe einen kleinen Test Programms. Es ist eine leicht angepasste Form des ccc\_hello\_world Beispiel Programms das mit der Pixy2 Arduino Library mitgeliefert kommt, da dieses genau das getan hat was ich testen wollte.

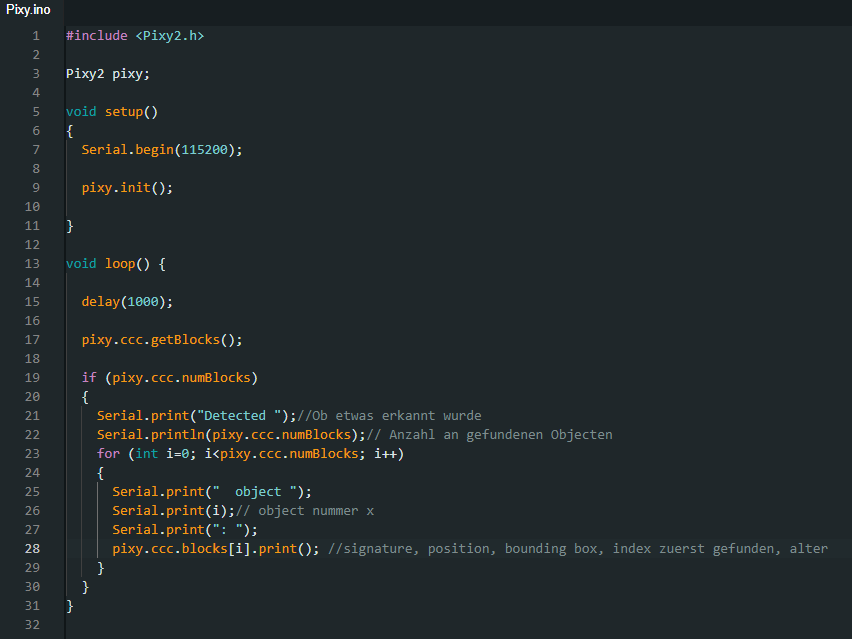


Abbildung 4 Screenshot meines Arduino Codes eigene Aufnahme

Indem Programm sieht man, dass das Pixy in der Pixy Library ein Object ist. Die Funktion pixy.ccc.getBlocks() aktualisiert die Variablen vom Pixy-Objekt mit den Daten die es vom Pixy bekommt. Dieses Programm gibt dann die Daten die die Pixy-Kamera dem Arduino sendet im Serial Port aus. Sodass wir sehen können ob das Pixy wirklich funktioniert und welche Daten es dem Arduino sendet.

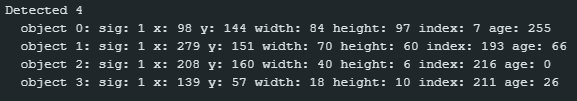


Abbildung 5 Screenshot Output Serial Port Arduino eigene Aufnahme

Wie man sieht schickt das Pixy dem Arduino eine Liste mit den Objekten die es erkannt hat. In der steht welches Objekt es meint erkannt zu haben mit sig kurz für Signatur. Die nächsten vier Werte geben die Koordinaten und Größe der Bounding-box des Objektes in Pixeln an und jedes Object bekommt einen Index zugeordnet mit dem es identifizierbar bar ist. Seit wie vielen Sekunden die Pixy2-Kamera ein Objekt bereits trackt wird mit dem age Wert angegeben. Mit diesen Werten könnte man dann anstatt sie nur anzuzeigen dann befehle für Motoren errechnen mit deren Hilfe es dann Beispielsweise ein Objekt verfolgen könnte.

## Wie Object-Detektion mit dem Pixy2 funktioniert

Die Objekterkennung der Pixy2-Kamera funktioniert mit einem Farbeerkennung-Algorithmus in Kombination mit einem region growing Algorithmus, mit dem er einfarbig zusammenhängende Objekte erkennt. Da das Pixy seine Objekterkennung mit normalen Algorithmen umsetzt handelt es sich hierbei bei nicht um eine KI.

## Problem und Performance der Pixy2-Kamera

Der Pixy Sensor hat große Problem mit den einfarbigen Objekten. Weile er alle Objekte mit der Farbe eines eingespeicherten Objekts als dieses Objekt identifiziert. Was vor allem wenn man zwei Objekte mit einer ähnlichen Farbe einspeichert. Er tut sich auch mit farblich komplexeren Objekten schwer diese wieder zu erkennen. Am besten funktionierte er mit Objekten oder Symbolen die aus wenigen klar unterscheidbaren Farben bestehen (Pixy, kein Datum).

Die Pixy-Kamera schafft es konstant mit über 60 Bilder pro Sekunde zu laufen womit es weniger als 16,7 Millisekunden pro Bild braucht. Dies erklärt sich damit, dass es nur eine Auflösung von 316 mal 208 Pixel besitzt und somit der Algorithmus nicht besonders viele Pixel verarbeiten muss.

# Tflite Model auf Raspberry Pi

Die zweite Option wie man künstliche Intelligenz auf Microkontrollern umsetzen kann die ich mir angeschaut habe, ist ein TensorFlow Lite Model auf einem Raspberry Pi zu verwenden. Der große Unterschied zu Variante mit dem Pixy ist, dass man hier ein im Vorhinein auf einem stärkeren Computer trainiertes KI-Modell verwendet, das auf bestimmte Objekte trainiert wurde. Das dann auf dem Raspberry Pi ausgeführt wird, in meinen Fall dem Raspberry Pi 4 Model B. Dies ist mit dem Raspberry Pi möglich da er im Vergleich zum Arduino deutlich mehr Leistung hat, sodass er in der Lage ist selbst Objekterkennung mit einem TFLite Modelle auszuführen.

Ein TFLite Modell kurz für TensorFlow Lite ist ein TensorFlow Model das für Hardware schwache System optimiert wurde und dadurch schneller ist und auf schwächere Hardware ausgeführt werden kann. TensorFlow ist eine von Google entwickeltes Framework für Maschinelles Lernen mit dem man Neuronale Netzwerke erstellen kann und es handelt sich um eine Plattform übergreifende Libary für künstliche Intelligenz (Dipl.-Ing. Luber & Litzel, 2018).

## TFLite auf Raspberry Pi installieren

Um TFLite Objekterkennungs-Modelle auf dem Raspberry Pi zu verwenden muss man zuerst TFLite auf dem Raspberry Pi installieren. Dafür sollte man seinen Raspberry Pi erstmal Updaten und auf die neuste Version bringen mit den Kommandozeile Befehlen: sudo apt-get update und sudo apt-get dist-upgrade, da ich meinen Raspberry Pi längere Zeit nicht verwendet hatte und er deshalb mehrere Updates runterladen musste dauerte das ganze bei mir fast eine Stunde. Der nächste Schritt ist das TensorFlow Lite Object Detection on Android and Raspberry Pi GitHub Project auf den Raspberry zu klonen und dann sollte man ein Python virtual Environment schaffen in dem man dann die von TFLite benötigten Libraries installiert. In Python ist ein virtuelles Environment ein vom Rest der Python Installation abgetrennter Teil, sodass alle Python Skripts die im virtualen Environment ausgeführt werden nur auf die Libraries im Environment zugreifen können (freeCodeCamp, 2022). Das ist wichtig, da man so mehrere Versionen derselben Python Library installieren kann. TFLite benötigt mehrere bestimmte Versionen von Python Libraries. So konnte ich sicherstellen das die Installation der Libraries die Tflite benötigt nicht mit meinen anderen Installationen in Konflikt gerät.

## TFLite Modelle auf Raspberry Pi verwenden

Ist TFLite erfolgreich auf dem Raspberry Pi aufgesetzt kann man nun entweder TFLite Objekterkennungs-Modelle aus dem Internet herunterladen oder eigene Modelle verwenden. Um dann die Modelle zu verwenden muss man eines der vier Skripte die in dem GitHub Projekt, das wir heruntergeladen haben sind, von der Kommandozeile aus ausführen. Der Name das Directory in dem sich das KI-Modell befindet, dass man verwenden will, muss mit einem Argument (--modeldir=Duck\_1000 s.u.) definiert werden. Man sollte dabei darauf achten in dem Virtuellen Environment zu sein das wir für TFLite erstellt haben, da es ansonsten nicht funktionieren wird. Der Befehl zum ausführen des TFLite\_detection\_webcam.py Skripts kann zum Beispiel so aussehen.

python TFLite\_detection\_webcam.py --modeldir=Duck\_1000

Das TFLite\_detection\_webcam.py Skript öffnet, wenn an den Raspberry Pi eine Kamera angeschlossen ist, ein Fenster in dem das letzte Bild der Kamera über das das Modell gelaufen ist. Die Verzögerung hierbei ist immens und die Framerate bleibt selbst mit dem Schnellsten Modellen im unteren einstelligen Bereich.

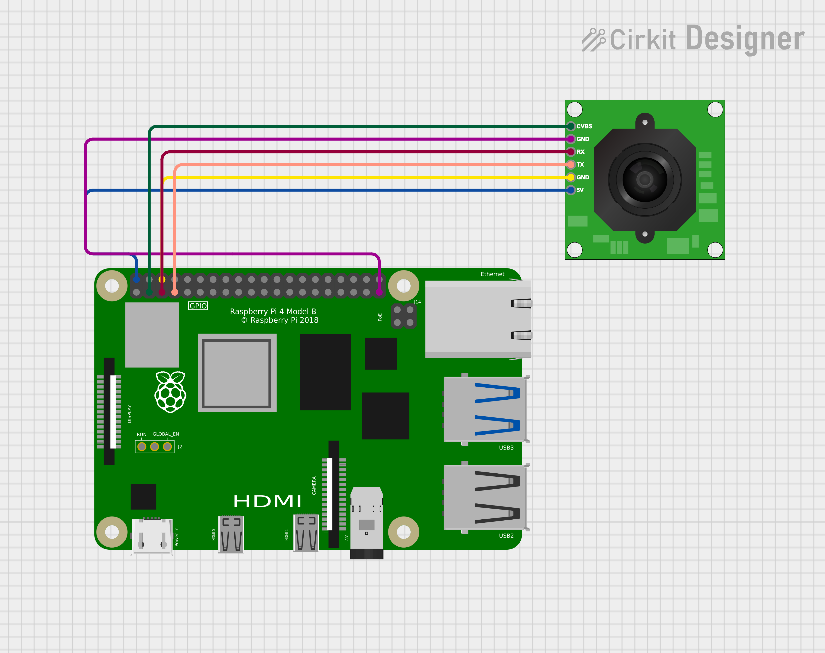


Abbildung 6 Schaltplan Raspberry Pi mit Kamera eigene Darstellung

Mit den drei anderen Scripts kann man das Modell über Bilder, ein Video oder einen Stream laufen lassen und bietet somit weitere Möglichkeiten der Nutzung. Die Abbildungen 9 und 10 in dieser Arbeit zum Beispiel habe ich mit dem TFLite\_detection\_image.py Skript erstellt.

## Eigenes Model im TFLite Format für den Raspberry Pi trainieren

Wie oben bereits angesprochen kann man natürlich auch selbst ein Eigenes Modell trainieren. Das bietet sich vor allem an wen man eine eigene Anwendung im Sinn hat. Es kann sein, dass es ihm Internet kein öffentlich verfügbares Object-Detektion Model gibt das auf die Objekte trainiert ist, die man mit der eigenen Anwendung erkenne will. Ich wollte schauen wie schwer es ist so ein Modell selbst zu erstellen und habe deshalb selbst ein Modell trainiert. Ich habe mich dafür entschieden mein Modell auf das Erkennen einer Stofftier Ente zu schulen. Dadurch dass es einfarbig ist und sich gut vom Hintergrund abhebt ist das für die KI ein leicht erkennbares Objekt. Ich habe mein Modell mit einem Pretrained TensorFlow Modell aus dem TensorFlow 2 Object Detection Model Zoo trainiert (Anhang 4). Ich habe festgestellt, dass ein eigenes TensorFlow Modell von Grund auf zu trainieren recht schwer ist. Nachdem ich die ersten paar Stunden eines über zehn stündigen Beginner Tutorials zu TensorFlow angeschaut hatte, habe ich mich gegen ein eigenes TensorFlow Modell entschieden.

### Trainings Daten erstellen

Um ein KI-Model zu trainieren braucht man ein Trainings Daten Set auf dem man es üben lassen kann. Um ein Objekterkennungs-Modell zu erstellen braucht man dafür gelabelte Bilder von den Objekten die es erkennen soll, in meinem Fall also von der Ente. Diese habe ich aus mehreren Perspektiven fotografiert. Dabei gilt es zu beachten, dass die Bilder die man macht, möglichst genau die Situationen widerspiegeln in denen das Modell dann später auch verwendet wird. Wenn das Modell später in verschieden Situation verwendet wird dann diese auch all in den Bildern vertreten sind. So wollte ich, dass es die Ente aus verschieden Blickwinkeln, Distanzen und unter unterschiedlichen Lichtverhältnissen erkennen kann. Deswegen habe ich 317 Bilder für mein erstes Modell aufgenommen.

Mit Fotos allein kann das Modell nicht trainiert werden. Deshalb mussten die 317 Fotos noch gelabelt werden. Dazu habe ich die open source Software Labelimg verwendet, mit der man Bilder labeln kann, indem man ein Kästchen um das Objekt zieht auf das man das Modell Trainieren will und das Objekt dann mit dem Label für das Objekt versieht.



Abbildung 7 Screenshot Labelimg eigene Aufnahme

Labelimg speichert dann diese Information in XML Files in PASCAL VOC Format, das ist ein Datei Format das ursprünglich für die Visual Object Challenge erschaffen wurde. Es ist ein Standard Format für Objekt-Detektion-Labels. Da es die Position eines Objektes in Form einer Bounding-Box die das Objekt eingrenzt speichert sowie dessen Label (roboflow, kein Datum). Beim Labeln der Bilder gilt zu beachten, dass die Bounding-Box die man um das Objekt macht möglichst eng ist aber das gesamte Objekt enthält. Der Prozess des Labelns hat etwas mehr als zwei Stunden in Anspruch genommen und war damit deutlich schneller als ich erwartet hatte.

### Versuch Model in Google Colab zu trainieren

Ich hatte ein Tutorial gefunden das mir zeigt wie man ein Objekterkennungs-Modell aus dem Model Zoo trainiert, zu dem ein jupyter Notebook auf Google Colab gehört Anhang 2. Ein Jupyter Notebook ist ein Datei Format das zur Python Library Jupyter gehört. Die erlaubt den Notebooks in denen man in einzelnen Code Blocks und Kommandozeilen Befehle auszuführen und sie im Browser auf einer lokal gehosteten Webseite zu bearbeiten (Bochkor & Krypczyk, 2022). Sie sind vor allem bei Projekten wie diesem Nützlich bei dem man zwischen Kommandozeile und Python ständig wechseln muss. Dieses Notebook konnte ich einfach in Colab ausführen, da es alle Python Skripts und Kommandozeile Befehle enthielt die für das Training des Modells notwendig sind. Deshalb war mein ursprünglicher Plan das Objekt-Detektion Modell einfach mit Hilfe dieses Tutorials und des Notebooks in Colab zu trainieren. Mit Google Colab kann jeder jupyter Notebooks im Browser auf einem Google Server der mit einer Grafikkarte ausgestattet ist ausführen lassen und ist auf Data Science und Maschinelles lernen ausgelegt (Google, kein Datum). Es hat auch anfangs gut funktioniert und ich konnte einfach dem Tutorial folgen. Ich musste nur meine Trainings Daten Hochladen und ein paar Variablen im Notebook an mein Modell anpassen. Als ich dann das Modell in Colab zum Schluss trainieren ließ wurde ich dann nach einer halben Stunde Training aus der Sitzung geschmissen, ohne dass mein Modell fertig trainiert wäre. Dabei habe ich allen Fortschritt des Trainings bis dahin verloren. Der Grund dafür lag darin das Colab ein Limit für die Tägliche Dauer der Nutzung von GPUs hat, das nicht genau spezifiziert ist, für Nutzer der gratis Version. Dieses Limit reichte aber nach mehreren Anläufen scheinbar nicht, um mein KI-Modell zu trainieren. Der Grund, dass das Tutorial dem ich gefolgt bin dies nicht erwähnt hatte lag daran das es von Anfang 2023 ist. Google hat seitdem die täglichen Limits für die gratis Version mehrfach verringert. Es hat also zum Zeitpunkt der Erstellung wahrscheinlich noch mit der gratis Version funktioniert.

### TensorFlow lokal aufsetzen

Da es nicht geklappt hat das KI-Modell auf Google colab zu Trainieren blieb mir nichts anderes übrig als, das KI-Modell lokal auf meiner eigenen Maschine zu trainieren. Um das zu bewerkstelligen musste ich TensorFlow auf meinem Computer aufsetzen. Dies erwies sich als schwieriger als erwartet. Nachdem ich mehreren längeren Tutorials gefolgt bin und kam immer wieder zu einem Punkt wo ich in eine Schleife geriet. Hierbei bekam ich eine Fehlermeldung bei der mir eine Libary sagt, dass sie eine andere Version von einer zweiten Libary braucht. Nur um dann nach einer Neuinstallation dieser Library festzustellen, dass eine dritte Libary wiederum eine andere Version dieser zweiten Libary braucht. Beim Versuch dieses Problem zu lösen fand ich heraus, dass jede Version von TensorFlow nur mit einer bestimmten Python, cuDNN und CUDA Versionen kompatible ist. Der Grund für mein Problem lag höchstwahrscheinlich daran, dass ich eine Falsche Python Version hatte und TensorFlow hat seit November 2022 den Grafikkarten Support für Windows eingestellt.

Da es keinen Sinn macht ein KI-Modell ohne eine Grafikkarte zu trainieren, entschied ich mich WSL auf meinem Computer zu installieren. WSL ist das Windows Subsystem für Linux mit WSL kann man ein Linux Environment parallel zum normalen Windows auf seinem Computer laufen lassen ohne eine Virtuelle Maschine zu benötigen (Microsoft, 2023). WSL zu installieren war recht einfach und ich brauchte nur den Kommando Zeil Befehl: wsl --install als Administrator ausführen.

Um das Problem der spezifischen Python Version anzugehen habe ich mir Anaconda in WSL installiert, um nicht immer, wenn ich feststellte, dass ich eine andere Python Version brauchte Python neu installieren zu müssen. Anaconda ist eine Python-Distribution mit der man Conda Environments erstellen kann. Diese Virtuelle Environments erlauben es einen aber auch eine Python Version festzulegen mit der alle Python Skripts ausgeführt werden während man in dem Conda Environment ist (Yegulalp, 2024).

Für CUDA und cuDNN gab es keine solche Option und ich musste sie mehrfach neu installieren. CUDA ist ein Toolkit das man braucht um auf NVIDIA Grafikkarten zu Entwickeln (NVIDIA, kein Datum). cuDNN ist die NVIDIA CUDA® Deep Neural Network library und ist dafür benötigt damit man KI-Modell auf der Grafikkarte mit CUDA Trainieren kann (NVIDIA, kein Datum). CUDA und cuDNN brauchte ich um meine Grafikkarte zum Trainieren verwenden zu können.

All dies erlaubte mir TensorFlow auf meinem Computer mit support für Training mit einer Grafikkarte zu installieren und ich entschied mich dem ursprünglichen Tutoriell mit dem Colab Notebook zu folgen, da es auch für Linux gedacht war, weil die Colab Server alle Linux als Betriebssystem haben und ich ja wusste, dass es funktioniert da ich mit ihm schon mal bis zum Training gekommen war. Ich habe dann die Kommandozeilenbefehle und die Python Teile im Notebook von den Colab Servern auf mein System angepasst. Somit konnte ich die Befehle direkt in der Kommandozeile von meiner WSL Installation ausführen und die Python Teile des Notebooks ihn einzelne Python Skripte übertragen. Das hat dann endlich funktioniert. Die Skripte befinden sich auch auf meinem GitHub erster Anhang.

### Parameter meines Modells

Natürlich musste ich auch beim trainieren meines Modells auf meinem Computer wieder die Parameter wie beim Training auf Colab meinem Modell anpassen. Diesmal zusätzlich auch noch ein paar weitere Parameter modifizieren damit mein Computer das Training bewältigen konnte. Zum einen musst ich ein config File erstellen indem ich hinterlege welches der Modelle aus dem TensorFlow 2 Object Detection Model Zoo (aus dem Internet bezogen) ich verwenden möchte. In meinem Fall war dies das „ssd\_mobilenet\_v2\_fpnlite\_320x320\_coco17\_tpu-8“ Modell. Die Anzahl an Schritten die ich Trainieren möchte definieren, während einem Schritt macht das Modell Vorhersagen und passt dann seine Parameter in Abhängigkeit von der Abweichung von der im Daten Set hinterlegten richtigen Antwort an, dem sogenannten Verlust. Die Batch Size bestimmt wie viele Bilder pro Trainings Schritt verwendet werden umso größer diese ist umso schneller wird das Modell trainiert. Aber umso höher die Batch Size desto mehr Grafikspeicher braucht es beim Training. Ich habe eine Batch Size von 8 verwendet. Ich habe mit 40.000 Schritten trainiert, welches die empfohlene Menge für das Modell war das ich gewählt hatte. Die Learning Rate ist eine Variable in der Funktion die bestimmt wie stark sich die Parameter in jedem Stepp verändern können und dafür sorgt, dass das Modell langsam konvergiert, dadurch dass sie während des Trainings immer kleiner wird.

### Training des Modells

Nach dem TensorFlow aufgesetzt war und alle Parameter eingestellt waren konnte ich dann mit dem Training des Modells beginnen. Das hat dann etwas weniger als zwei Stunden gebraucht, da ich einen relativ guten Computer und vor allem eine gute Grafikarte mit viel Grafikartenspeicher habe (GeForce RTX 3090 mit 24 GB VRAM). Während des Trainings lief mein Computer fast durchgängig auf hundert Prozent seiner Leistung.

Denn Trainings Prozess konnte ich über Tensorboard während des Trainings überwachen. Tensorboard ist ein Visualisierungstool von Google das man mit TensorFlow verwenden kann. Die Entwicklung des Modells während des Trainings zu sehen ist wichtig, denn sollten irgendwelche Einstellungen nicht stimmen, sieht man das dort direkt. So ist bei mir der Loss beim ersten Trainingsversuch nach kurzer Zeit nicht mehr weiter gesunken, sondern immer größer geworden, da ich die Learning Rate falsch eingestellt hatte. Zwei weitere Trainings Versuche scheiterten daran das meine GPU nach kurzer Zeit der Grafikspeicher ausgegangen ist, da ich die Batch Size zu groß eingestellt hatte. Deshalb musste ich meine Batch Size von ursprünglich 16 auf 8 verringern.

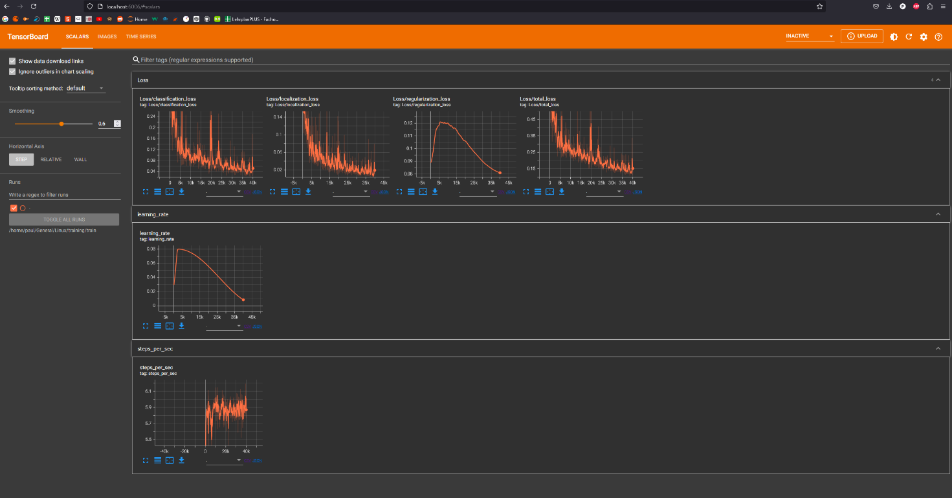


Abbildung 8 Screenshot von Tensorboard nach Ende des Trainings von Duck\_317 eigene Aufnahme

Auf Tensorboard kann man von rechts nach links den Klassifizierungsverlust, den Lokalisationsverlust, den Regulierungsverlust und den Totalverlust sehen. Verlust ist bei Objekterkennungs-Modellen ein Weg die Intensität ihrer Fehler zu messen. Indem man die Abweichung der erzielten Ergebnisse von den gewünschten Ergebnissen berechnet. Der Klassifizierungsverlust gibt die Abweichung bei der Klassifizierung der Objekte an. Der Lokalisation Verlust die Abweichung der von der KI erkannten Bounding-Boxes von den richtigen Bounding-Boxen (AMBOLT AI, kein Datum). Der Regulierungsverlust ist ein Verlust der dem Totalverlust einfach hinzugefügt wird. Der Regulierungsverlust ist hierbei eine Weg Overfitting zu vermeiden was das Phänomen beschreibt, wenn sich ein KI-Modell zu sehr auf seine Trainings Daten einschießt und dadurch nicht in der Lage ist sich Situationen Anzupassen die nicht in seinem Trainings Set enthalten waren (Simplilearn, 2024). In unserem Beispiel könnte es dann vorkommen, dass es die Ente vor einem Hintergrund der nicht im den Trainings Daten vorkommt nicht erkennt. Der Totalverlust ist wird aus allen anderen Verlusten addiert. Unter den Verlusten befindet sich die Learning Rate und ganz unten wie viele Schritte mein Computer pro Sekunde schafft.

Zu beobachten wie sich das Modell während des Trainings entwickelt ist auch wichtig, um zu entscheiden ob man das Training frühzeitig beenden kann, wenn sich der Verlust nicht mehr Verringert. Sprich das Modell konvergiert oder um zu sehen das sich der Verlust bis zum Ende verringert hat, um dann Entscheiden zu können das Modell noch weiter Trainieren zu lassen. Bei mir hat es mit den Eingestellten 40000 Stepps genau gepasst.

### Model zu Tflite Format konvertieren

Das Model das ich dann trainiert hatte konnte ich nicht direkt auf den Raspberry Pi verwenden, da es bisher noch ein normales TensorFlow-Modell war. Also war der nächste Schritt das Modell zu einem Tflite Model zu konvertieren. Damit hatte ich dann mein erstes Objekt-Detektion Modell Duck\_317. Es ist auf meinem GitHub zu finden zusammen mit meinen anderen Modellen im TFLite\_Modelle Ordner Anhang 1. Die Konvertierung selbst ist recht uninteressant. Man führt ein Python Skript aus indem man ein Konverter Objekt erstellt und eine Methode des Konverter Objekts ausführt.

## Model mit Quantisierung optimieren

Man kann nun nach dem Training mit dem fertigen Modell noch eine Quantisierung durchführen. Dabei werden die Variablen der Gewichte des Modells von einem genaueren größeren Datentyp in einem kleineren ungenaueren umgewandelt (zum Beispiel von uint16 zu uint8). Das verringert die Modelgröße und verbessert dessen Geschwindigkeit. Es verliert dabei aber auch an Präzision (TensorFlow, 2022). Ich habe mein Model auch quantifiziert da ich möglichst die beste Performance rausholen wollte und um herauszufinden wie sich das auf das Modell auswirkt. Am Ende hatte ich dann auch noch zusätzlich eine quantifizierte Version meines Modells Duck\_317\_quant.

## Performance des selbst trainierten KI-Modells

Nachdem ich dann das KI-Modell und seine quantifizierte Version auf den Raspberry Pi übertragen hatte, habe ich beide dort ihre Performance gemessen und sonstige Probleme festgestellt.

Ein Problem des Modells war die Tatsache, dass es alle möglichen gelben Objekte als Ente erkannte. Besonders ein anderes Kuscheltier eine gelbe Banane hat es des Öfteren mit der Ente verwechselt. Dieses Problem wurde bei größeren Entfernungen und unter schlechten Lichtbedingungen schlimmer.

Was ihre Performance angeht so gibt es zwei wichtige Maßeinheiten mit der man Objekterkennungs-Modell normalerweise beschreibt. Einmal wie schnell sie sind und ihren mAP.

### Der mAP meines Modells

Der mAP oder mean average precision ist eine Maßeinheit mit der die Präzision von Objekterkennungs-Modellen verglichen wird. Das mean steht dabei für den Durchschnitt der average precision kurz AP über mehrere Klassen dar. Mein Modell hat nur eine Klasse. Deshalb entspricht der mAP gleich dem AP der Klasse der Ente. Wie sich der AP im Detail zusammensetzt kann man im Artikel der im Anhang 3 verlinkt ist nachlesen. Sehr vereinfacht gesagt, setzt er sich aus der Abweichung der Bounding-Box (Viereck welches das Objekt eingrenzt) der Vorhersage des Modells zur eigentlichen Bounding-Box und der Sicherheit und Akkuratheit mit der das Modell die Klasse erkennt zusammen. Damit errechnet er dann einen Wert der zwischen null und eins liegt, wobei eins perfekt wäre. Also mit einem Wert von 1 macht das Modell alles richtig, dass ist aber realistisch nicht erreichbar. Der mAP wird normalerweise nicht zwischen eins und Null angegeben, sondern in Prozent.

In dem Notebook des Tutorials war auch ein Python Skript mit dem man den mAP seines Modells berechnen lassen konnte. Der mAP meines Standard Modells Duck\_317 betrug 53,20% und der der quantifizierten Version 30,11%, was nicht wirklich besonders gute Werte sind. Hierbei gilt zu erwähnen, dass der die Werte des mAP abweicht, wenn man die Gelabelten Bilder austauscht. Alle mAP Werte die ich errechnet habe sind mit denselben Bildern errechnet worden um vergleichbar zu sein.

### Die Geschwindigkeit meines Modells

Um die Geschwindigkeit der Modelle zu messen habe ich mir von ChatGPT ein Python Skript erstellen lassen, dass die Modelle mit dummy Daten füttert und dann beobachtete ich wie lange sie brauchen um zu einem Ergebnis zu kommen. Duck\_317 hat ca. 318 Millisekunden gebraucht und Duck\_317\_quant hat es in ca. 234 Millisekunden geschafft. Bei jeder Ausführung des Skripts haben sich leichte Abweichungen ergeben.

## Verbesserung des Modells durch neu Training mit mehr Bildern

Da ich nicht mit der Akkuratheit meines Models zufrieden war und ich schauen wollte wie sich die Perfomance des Modells verändert, wenn ich es mit weiteren Bildern trainiere habe ich noch 683 weitere Bilder gemacht und Gelabelt. Mit diesen habe ich zwei weiteren Modellen trainiert, eins mit 650 Bildern Duck\_650 und eins mit 1000 Bildern Duck\_1000 die ich auch jeweils quantifiziert habe.

Um das Problem das Duck\_317 viele gelben Objekte als Ente erkennt anzugehen habe ich bei den Bildern für die späteren Modellen bewusst Bilder von der Ente gemacht auf denen auch andere gelbe Objekte wie die Banane zu sehen waren. Damit es lernt das Gelb nicht automatisch Ente bedeutet. Das hat auch funktioniert die späteren Modelle konnten deutlich besser zwischen der Ente und anderen gelben Objekten unterscheiden aber auch nicht immer.

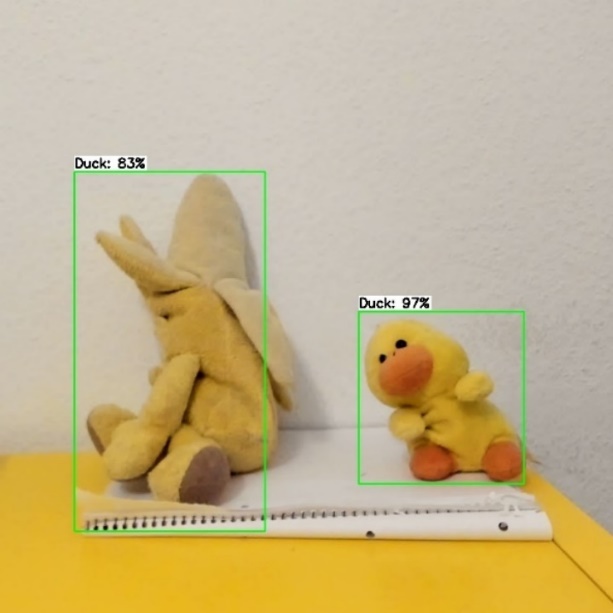
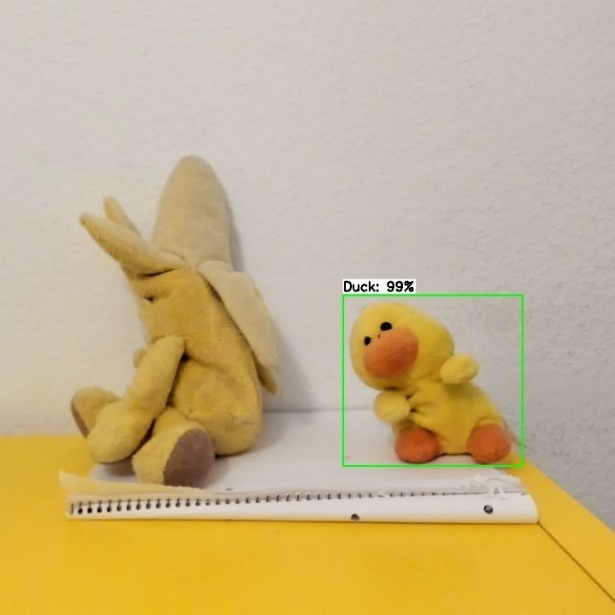
 

Abbildung 9 Bild über das Duck\_317 gelaufen ist eigene Aufnahme

Abbildung 10 Bild über das Duck\_1000 gelaufen ist eigene Aufnahme

Beim Training der Modelle mit mehr Bilder habe ich festgestellt, dass diese deutlich länger im Training brauchten bis sie alle 40000 Stepps abgeschlossen hatten. Das Modell mit 650 Bildern hat zwei Stunden und eine halbe gebraucht. Das Modell mit 1000 Bildern fast drei Stunden.

Was ihre Geschwindigkeit anging waren sie und ihre quantifizierten Versionen nochmal deutlich langsamer als die Modelle mit nur 317 Bildern.

Abbildung 11 Grafik Geschwindigkeit eigene Darstellung

Interessant ist hier, dass die Unterschiede in der Geschwindigkeit zwischen den Modellen mit 650 und 1000 winzig sind.

Beim mAP gibt es dann deutlich größere Unterschiede zwischen den einzelnen Modellen und mein Duck\_1000 Modell schafft sogar fast einen mAP von 80% zu erreichen, was eine deutliche Verbesserung zum Duck\_317 Modell darstellt.

Abbildung 12 Grafik mAP eigene Darstellung

Hier sieht man auch nochmal besonders deutlich den Verlust in Präzision zwischen Quantifizierten und nicht Quantifizierten Modellen.

# Fazit

Objekterkennung auf Microcontrollern lässt sich sowohl mit der Pixy2-Kamera und Arduino als auch auf dem Raspberry Pi mit einem TFLite Modell umsetzen, wen man grundlegende Programmierkenntnisse besitzt. Wobei es mit dem Pixy zwei deutlich einfacher und schneller ist. Wenn man die richtige Version bestellt muss nur die Arduino Library heruntergeladen werden. Während allein die Einrichtung von TFLite auf dem Raspberry Pi mehr als eine Stunde einnehmen kann und auch deutlich komplizierter ist. Wenn man dann noch selbst ein KI-Modell trainiert benötigte man sehr viel Zeit, vor allem wenn man zum ersten Mal ein KI-Modell trainiert.

Dafür ist das Modell dann aber auch deutlich Präziser als die Pixy2-Kamera und kann auf jedes Objekt trainiert werden. Im Gegensatz zum Pixy das Objekte mit einer sehr einfachen Einfärbung benötigt. Was Geschwindigkeit angeht ist das Pixy deutlich schneller als TFLite-Modelle.

Daraus folgerte ich, dass die Pixy2-Kamera deutlich besser für Projekte geeignet ist die wenig Latenz verlangen wie zu Beispiel bei einem Roboter der Hindernisse ausweichen soll. Während die TFLite-Modelle auf dem Raspberry besser für Projekte geeignet sind bei denen ein Objekt klar identifiziert werden muss uns längere Verzögerungen keine Rolle spielen. Beispielsweise einer smarten Tür die erkenne soll, ob die Person die vor ihr steht zum Haushalt gehört und reindarf.

Es lässt sich daraus auch meiner Meinung nach schließen, dass es noch etwas dauern wird bis wir gute schnelle KI-Modelle auf Hardware schwachen Geräten haben.

# Literaturverzeichnis

AMBOLT AI. (kein Datum). *How Object Detectors learn*. Abgerufen am 12. 1 2025 von https://ambolt.io/en/how-object-detectors-learn/

Bochkor, E., & Krypczyk, V. (7 2022). *Jupyter Notebook: Python und mehr!* Abgerufen am 12. 1 2025 von https://entwickler.de/datenbanken/einfuehrung-jupyter-notebook

Dipl.-Ing. Luber, S., & Litzel, N. (7. 2 2018). *Was ist TensorFlow?* Abgerufen am 12. 1 2025 von https://www.bigdata-insider.de/was-ist-tensorflow-a-684272/

freeCodeCamp. (11. 4 2022). *How to Set Up a Virtual Environment in Python – And Why It's Useful*. Abgerufen am 12. 1 2025 von https://www.freecodecamp.org/news/how-to-setup-virtual-environments-in-python/

Google. (kein Datum). *Willkommen bei Colab!* Abgerufen am 12. 1 2025 von https://colab.research.google.com/?hl=de

MathWorks. (kein Datum). *What Is Object Detection?* Abgerufen am 12. 1 2025 von https://de.mathworks.com/discovery/object-detection.html

Microsoft. (11. 29 2023). *What is the Windows Subsystem for Linux?* Abgerufen am 12. 1 2025 von https://learn.microsoft.com/en-us/windows/wsl/about

NVIDIA. (kein Datum). *CUDA Toolkit*. Abgerufen am 12. 1 2025 von https://developer.nvidia.com/cuda-toolkit

NVIDIA. (kein Datum). *NVIDIA cuDNN*. Abgerufen am 12. 1 2025 von https://developer.nvidia.com/cudnn

Pixy. (kein Datum). *Teach Pixy2 an Object*. Abgerufen am 12. 1 2025 von https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:teach\_pixy\_an\_object\_2

roboflow. (kein Datum). *Pascal VOC XML*. Abgerufen am 12. 1 2025 von https://roboflow.com/formats/pascal-voc-xml

Simplilearn. (15. 5 2024). *The Best Guide to Regularization in Machine Learning*. Abgerufen am 12. 1 2025 von https://www.simplilearn.com/tutorials/machine-learning-tutorial/regularization-in-machine-learning

TensorFlow. (8. 3 2022). *Post-training quantization*. Abgerufen am 12. 1 2025 von https://www.tensorflow.org/model\_optimization/guide/quantization/post\_training

Yegulalp, S. (23. 12 2024). *Anaconda Python erklärt*. Abgerufen am 12. 1 2025 von https://www.computerwoche.de/article/2831836/anaconda-python-erklaert.html

# Abbildungsverzeichnis

[Abbildung 1 Bild vom Pixy2 eigene Aufnahme 3](#_Toc187740159)

[Abbildung 2 Screenshot von PixyMon eigene Aufnahme 4](#_Toc187740160)

[Abbildung 3 Schaltplan Arduin mit Pixy eigene Darstellung 5](#_Toc187740161)

[Abbildung 4 Screenshot meines Arduino Codes eigene Aufnahme 5](#_Toc187740162)

[Abbildung 5 Screenshot Output Serial Port Arduino eigene Aufnahme 6](#_Toc187740163)

[Abbildung 6 Schaltplan Raspberry Pi mit Kamera eigene Darstellung 8](#_Toc187740164)

[Abbildung 7 Screenshot Labelimg eigene Aufnahme 9](#_Toc187740165)

[Abbildung 8 Screenshot von Tensorboard nach Ende des Trainings von Duck\_317 eigene Aufnahme 13](#_Toc187740166)

[Abbildung 9 Bild über das Duck\_317 gelaufen ist eigene Aufnahme 16](#_Toc187740167)

[Abbildung 10 Bild über das Duck\_1000 gelaufen ist eigene Aufnahme 16](#_Toc187740168)

[Abbildung 11 Grafik Geschwindigkeit eigene Darstellung 16](#_Toc187740169)

[Abbildung 12 Grafik mAP eigene Darstellung 17](#_Toc187740170)

# KI-Verzeichnis

|  |  |  |  |
| --- | --- | --- | --- |
| **KI:** | **Befehl:** | **Ausgabe:** | **Datum:** |
| ChatGPT | How to test speed TensorFlow lite model? | Vorschlag der nicht auf dem Raspberry funktioniert hat TensorFlow Lite Benchmark Tool zu verwenden. Sowie  mehrere kürzere Python Skripts mit denen man es selbst testen kann. Die ich zu einem großem zusammengefügt habe. | 21.09.24 |

# Anhang

1. GitHub: <https://github.com/Luap25/Seminararbeit>
2. Colab Notebook: <https://colab.research.google.com/github/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/Train_TFLite2_Object_Detction_Model.ipynb#scrollTo=kPg8oMnQDYKl>
3. Artikel über mAP: <https://blog.roboflow.com/mean-average-precision/>
4. TensorFlow 2 Detection Model Zoo: https://github.com/tensorflow/models/blob/master/research/object\_detection/g3doc/tf2\_detection\_zoo.md

# Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Seminararbeit selbständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.



München, 13.01.25 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ort, Datum Unterschrift