**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

[Inhalt 1](#_Toc187703292)

[1 Einleitung 1](#_Toc187703293)

[2 Pixy mit Arduino 3](#_Toc187703294)

[2.1 Anschaffung 3](#_Toc187703295)

[2.2 Einrichten 3](#_Toc187703296)

[2.3 Pixy2 mit dem Computer verbinden 3](#_Toc187703297)

[2.4 Pixy mit Arduino verwenden 4](#_Toc187703298)

[2.5 Arduino Test Code 4](#_Toc187703299)

[2.6 Wie Object-Detektion mit dem Pixy2 funktioniert 4](#_Toc187703300)

[2.7 Problem des Pixys 4](#_Toc187703301)

[2.8 Weitere Anwendungen des Pixys Außer Objekt-Detektion 4](#_Toc187703302)

[3 Tflite Model auf Raspberry Pi 5](#_Toc187703303)

[3.1 TFLite auf Raspberry Pi installieren 5](#_Toc187703304)

[3.2 TFLite Modelle auf Raspberry Pi verwenden 5](#_Toc187703305)

[3.3 Eigenes Model im TFLite Format für den Raspberry Pi trainieren 6](#_Toc187703306)

[3.3.1 Trainings Daten erstellen 6](#_Toc187703307)

[3.3.2 Versuch Model in Google Colab zu trainieren 7](#_Toc187703308)

[3.3.3 TensorFlow lokal aufsetzen 7](#_Toc187703309)

[3.3.4 Parameter meines Modells 8](#_Toc187703310)

[3.3.5 Training des Modells 9](#_Toc187703311)

[3.3.6 Model zu Tflite Format konvertieren 10](#_Toc187703312)

[3.4 Model mit Quantisierung optimieren 10](#_Toc187703313)

[3.5 Performance des selbst trainierten KI-Modells 10](#_Toc187703314)

[3.5.1 Der mAP meines Modells 11](#_Toc187703315)

[3.5.2 Die Geschwindigkeit meines Modells 11](#_Toc187703316)

[3.6 Verbesserung des Modells durch neu Training 11](#_Toc187703317)

[4 Ergebnis 12](#_Toc187703318)

[5 Fazit 13](#_Toc187703319)

[6 Literaturverzeichnis 14](#_Toc187703320)

[7 Abbildungsverzeichnis 15](#_Toc187703321)

[8 KI-Verzeichnis 16](#_Toc187703322)

[9 Anhang 17](#_Toc187703323)

[10 Eidesstattliche Erklärung 18](#_Toc187703324)

# 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 auch in den nächsten Jahren einen immer weiteren Einzug in unser Leben finden. 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 Objekt Detektion. Objekt Detektion ist eine Computer-Vision-Technik die Objekte in einem Video oder Bild lokalisiert und erkennt. Objekt-Detektion-Algorithmen benutzen 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 Objekte Detektion 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 schon jetzt lassen sich mit Objekt-Detektion viele interessante Anwendungen umsetzen.

Das Problem mit KI auf Microkontrollern ist das sie durch ihre stark begrenzte Hardware Leistung Probleme damit haben die Leistung intensiven KI-Anwendungen auszuführen. Will man Object-Detektion Modell in Echtzeit ausführen kommt man dann schnell an Grenzen. Zwar werden Prozessoren immer Leistung fähiger, kleiner und billiger [Aber]. Deshalb und weil diese Geräte Preisewert sein müssen werden sie nie die Fähigkeit haben die momentan besten KI-Modelle auszuführen, da diese zu viel Rechenleistung benötigen.

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 Objekt Detektion 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 Edge Devices wie Microcontroller Laufen können.

Edge-Devices oder auf Deutsch Edge-Geräte sind Geräte die an der Grenze von zwei Netzwerken liegen und dienen als eingangs oder Ausgangs punkte des Netzwerks. Vor allem im Rahmen des Internets der Dinge sind Edge-Devices wichtig da sie die Geräte sind die beim Nutzer stehen.

Deshalb werde ich mir in dieser Arbeit angucken wie gut man Objekt Detektion auf Microcontrollern umsetzen kann und wie deren Performance ist. An zwei konkreten Beispielen einmal dem Pixy2 smart Vision Sensor mit einem Arduino und einem TFLite Modellen auf einem Raspberry Pi mit Kamera.

# Pixy mit Arduino

Eine der Varianten die ich mir angeschaut habe um Objekt Detektion in DIY-Projekten umzusetzen war eine out of the Box Solution der Pixy2 Sensor in Kombination mit einem Arduino. Pixy ist ein smart Vision Sensor mit dessen Hilfe man Objekt-Detektion . Da die Berechnungen nicht auf dem Arduino stattfinden ist es egal welchen Arduino man nimmt ich habe meinen Aufbau mit dem Arduino Nano und dem Arduino uno R4 WIFI getestet.

## Anschaffung

Der Pixy smart Vision Sensor ist im Vergleich zu anderen Sensoren recht teuer da er einen eigen kleinen Prozessor besitzt. Den er auch braucht um das Objekt-Detektion 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 ist. Beim Kaufe 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, 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 halten und konnte dann die neue Firmware aufspielen.

## Pixy2 mit dem Computer verbinden

Um den Pixy2 Sensor an einem Computer zu benutzen kann man es einfach 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.

[Bild Pixy Mon]

In PixyMon kann man auch Objekte einspeichern die der Sensor erkennen soll bis zu vierzehn Stück. Wobei sieben davon einfarbige Objekte seihen müssen und die anderen sieben aus mehreren Farben bestehen müssen. 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 funktioniert immer auch wenn die Pixy-Kamer nicht mit dem Computer verbunden ist.

## 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 Anschlüssen des Arduinos über den es mit dem Arduino Verbunden ist zusammengehören. Mit diesem selbst gebauten Kabel konnte ich dann die Pixy-Kamera mit dem Arduino verbinden.

[Bild Schaltplan]

Um mit dem Pixy dann auch kommunizieren zu können musste ich auch die Arduino Pixy2 Library in der Arduino IDE installieren [Arduin IDE erklären?]. 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. [Anhang?]

### Arduino Test Code

Ich habe dann getestet ob der Arduino mit dem Pixy auch funktioniert mithilfe einen kleinen Test Das Programm ist eine 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.

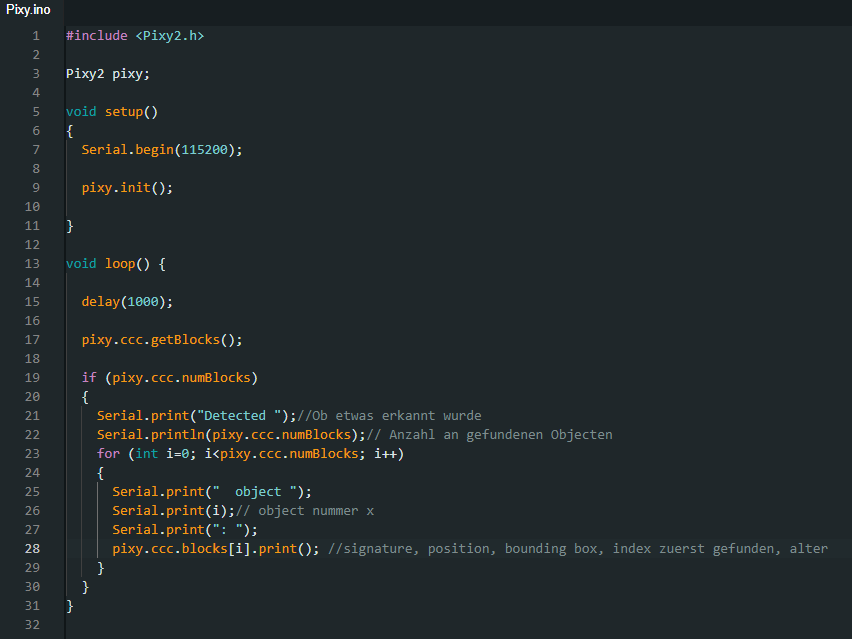


Abbildung 1 Screenshot meines Arduino Codes

Dieses Programm gibt das was die Pixy-Kamera dem Arduino sendet im Serial Port aus. Sodass wir sehen können ob das Pixy

[Bild Arduino Serial]

Wie man sieht schickt das Pixy dem Arduino mehrmals die Sekunde eine List mit den Objekten die es erkannt hat. In der steht welches Objekt es erkannt hat mit sig. Die nächsten vier Werte geben die Koordinaten und Größe der Bounding box des Objektes in Pixeln an und jedes Object bekommt . .Mit diesen Werten könnte man dann

## Wie Object-Detektion mit dem Pixy2 funktioniert

Die Pixy2-Kamera

## Problem der Pixy2-Kamera

Der Pixy Sensor hat große Problem Objekte mit ähnlicher Farbe auseinander zu halten und kann gleiche Objekte unter unterschiedlichen Licht Bedingungen nicht auseinanderhalten. 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.

## Weitere Anwendungen des Pixys Außer Objekt-Detektion

Der Pixy Vision Sensor besitzt zwei Programme einmal das Color connected components program dass wir für Object-Detektion verwenden und das Line Tracking Programm. Das dafür da ist Linien zu erkennen

# 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 das man hier ein im Vorhinein auf einem stärkeren Computer trainiertes KI-Modell verwendet, das dann auf dem Raspberry Pi ausgeführt wird. Das auf bestimmte Objekte trainiert wurde und diese erkennt anstatt wie beim Pixy, wo Objekte nur kurz vorgehalten werden müssen. Dies ist mit dem Raspberry Pi möglich da er im Vergleich zum Arduino deutlich mehr Leistung hat, sodass er in der Lage ist selbst einfache TFLite Modelle auszuführen.

## TFLite auf Raspberry Pi installieren

Um TFLite Objekt-Detektion Modelle auf dem Raspberry Pi zu verwenden muss man erst TFLite auf dem Raspberry Pi installieren. Dafür sollte man seinen Raspberry Pi erstmal Updaten und auf die neuste Version bringen mit den 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 Tflite GitHub Project auf den Raspberry zu klonen und dann ein Python virtual Environment zu schaffen in dem man dann die von TFLite benötigten Libaries installiert. In Python ist virtuell Environment ein vom Rest der Python Installation abgetrennter Teil, sodass alle python scripts die im virtualen Environment ausgeführt werden nur auf die Libaries im Environment zugreifen können (freeCodeCamp, 2022). Das ist wichtig da man so mehrere Versionen derselben Libarie installiert haben kann so braucht TFLite bestimmte Versionen von Libaries und so konnte ich sicherstellen das die Installation der Libaries 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 Objekt-Detektion Modelle aus dem Internet herunterladen oder eigene Modelle verwenden . Um dann das Modell zu verwenden muss man eines der vier scripte die in dem GitHub Projekt das wir heruntergeladen haben vom Shell aus ausführen lassen und mindestens ein Argument passen nämlich der Name das Directory in dem sich das KI-Modell, dass man verwenden will befindet. Man sollte dabei darauf achten in dem Virtuellen Environment zu sein das wir für TFLite erstellt haben.

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

Das TFLite\_detection\_webcam.py script öffnet, wenn an den Raspberry Pi eine Kamera angeschlossen ist ein Fenster in dem den letzten Frame über den das Programm gelaufen ist Angezeigt wird. Es ist das script

[Bild Webcam stream]

Mit den drei anderen Scripts kann man das Modell über ein oder mehrere Bilder, ein Video oder einen Stream laufen lassen und haben auch ihren Nutzen. Die Bilder in dieser Arbeit zum Beispiel auf denen Bounding boxes zu sehen sind habe ich alle mit dem TFLite\_detection\_image.py script erstellt.

[Aufbau Raspberry Camera]

## 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 Eigen Anwendung im Sinn hat da es sein kann das 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 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 trainieren, da es noch kein Modell gibt das auf diese Ente trainiert ist und es dadurch, dass es einfarbig ist und sich gut vom Hintergrund abhebt für die KI kein allzu schwieriges Objekt darstellte. Ich habe mein Modell mit einem Pretrained TensorFlow Modell aus dem TensorFlow 2 Object Detection Model Zoo trainiert, da ich festgestellt habe das ein eigen TensorFlow Modell von Grund auf zu trainieren recht schwer ist und meine momentanen Fähigkeiten übersteigt nach dem ich die ersten paar Stunden eines über zehn stündigen Beginner Tutorials zu TensorFlow angeschaut hatte. Der TensorFlow 2 Object Detection Model Zoo ist eine Ansammlung von Objekt-Detektion Modells

### Trainings Daten erstellen

Um ein KI-Model zu trainieren braucht man Trainings Daten auf denen man es trainieren kann. Um ein Objekt-Detektion Modell zu erstellen braucht man dafür gelabelte Bilder von dem Objekt das es erkennen soll, in meinem Fall von der Ente. Also musste ich Bilder von der Ente machen auf denen ich das Model dann trainieren kann. Dabei gilt zu beachten das die Bilder die man macht möglichst genau die Situationen widerspiegeln in denen das Modell dann später auch verwendet wird, so dass wenn das Modell später in verschieden Situation verwendet wird diese auch all in den Bildern vertreten sind. So wollte ich das es die Ente aus verschieden Blickwinkeln, Distanzen und unter unterschiedlichen Lichtverhältnissen erkennen kann. Deswegen habe ich Bilder gemacht die alle diese verschiedenen Situationen widerspiegeln. Insgesamt 317 Stück für mein einfaches erstes Modell.

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.

[Bild Labelimg]

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 und ist ein Standard Format für Objekt-Detektion-Labels da es die Position eines Objektes in Form eines Vierecks, dass das Objekt eingrenzt speichert sowie dessen Label (roboflow, kein Datum). Beim Labeln der Bilder gilt zu beachten, dass das Kästchen das 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.

### Versuch Model in Google Colab zu trainieren

Da ich ein Tutorial gefunden hatte zu dem ein jupyter Notebook auf Google Colab gehört. [juypiter]. Dies hätte ich einfach in Colab ausführen können, da es alle Python Skripts enthielt die Für das Training des Modells notwendig sind. 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 angelegt (Google, kein Datum). Es hat auch anfangs gut funktioniert ich konnte einfach dem Tutorial folgen und 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 das Mein Modell fertig Trainiert wäre und habe dabei allen Fortschritt im Training 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 reicht 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 und Google seitdem die Täglichen Limits für die gratis Version mehrfach verringert hat. 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 immer wieder an einen Punkt kam wo ich in eine Schleife geriet bei der ich eine Fehlermeldung bekam bei der mir eine Libary sagt das sie eine andere Version von einer zweiten Libary braucht nur um dann festzustellen das 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 und ich das beachten muss, sowie das TensorFlow seit November 2022 den support für Graphikkarten auf Windows eingestellt hat.

Da es keinen Sinn macht ein KI-Modell ohne eine Grafikkarte zu trainieren da das ewig dauern würde. 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). Das ist recht einfach man braucht nur den Befehl: wsl --install als Administrator ausführen.

Um das Problem der spezifischen Python Version anzugehen habe ich mir Anaconda installiert, um nicht immer, wenn ich feststelle das ich eine andere Python Version brauch Python neu installieren zu müssen. Anaconda ist eine Python-Distribution mit der man Conda Environments erstellen kann diese sind wie 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).

Dies erlaubte mir TensorFlow auf meinem Computer mit support für Training auf einer Grafikkarte zu installieren und ich entschied mich dem Ursprünglichen tutoriell mit dem Colab Notebook zu folgen, da es auch für Linux gedacht war da 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. Und dann die Befehle direkt im Shell von meiner WSL Installation ausgeführt und die Python teile des Notebooks ihn einzelne Python Skripte übertragen und auch vom Shell aus ausgeführt. Das hat dann endlich funktioniert.

### 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 und diesmal zusätzlich auch noch ein paar weitere Parameter anpassen damit mein Computer das Training bewältigen kann. Zum einen musst ich ein config File erstellen indem ich hinterlege welches der Modelle aus dem TensorFlow 2 Object Detection Model Zoo ich heruntergeladen habe und verwenden möchte in meinem Fall ssd\_mobilenet\_v2\_fpnlite\_320x320\_coco17\_tpu-8. Die Anzahl an Schritten die ich Trainieren möchte, 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. Die Batch Size die Batch Size bestimmt wie viele Bilder pro Trainings Schritt verwendet werden umso größer umso schneller wird das Modell trainiert aber umso mehr Grafikspeicher braucht es beim Training, ich habe eine Batch Size von 8 verwendet. Die Anzahl an schritten bis es einmal den Gesamte Datensatz gesehen hat berechnet sich aus Größe des Datensatz durch Batchsize in meinem Fall also hat das Modell den Gesamten Datensatz nach 40 Schritten zum ersten Mal komplett gesehen und fängt wieder von vorne an. Ich habe mit 40000 Schritten trainiert, da das die Empfohlene Menge für das Modell war das ich gewählt hatte. Damit hat es sich dann den gesamten Datensatz 1000-mal angeschaut.

### Training des Modells

Nach dem TensorFlow aufgesetzt war und alle Parameter eingestellt waren. Konnte ich dann mit dem Training des Modells beginne. Das hat dann etwas weniger als zwei Stunden gebraucht, da ich einen relativ guten Computer und vor allem eine gute Grafikarte mit viel Grafikartenspeicher hatte mit der GeForce RTX 3090 mit 24 GB. Während des Trainingslief 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 Trainings Versuch 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 16 auf 8 verringern.

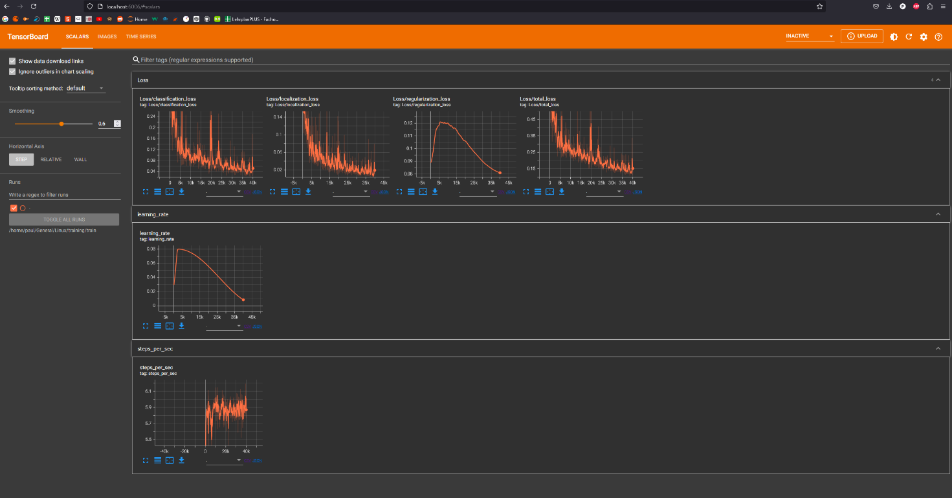


Abbildung 2 Screenshot von Tensorboard nach Ende des Trainings von Duck\_317

Auf Tensorboard kann man von recht nach links den Klassifizierung Verlust, den Lokalisation Verlust, den Regulierung Verlust und den Totalen Verlust sehen. Verlust ist bei Objekt-Detektion Modellen ein weg die Intensität ihrer Fehler zu Messen.

Der Klassifizierung Verlust gibt an

Der Lokalisations Verlust

Der Regulierung Verlust gib an welchem Teil des Totalen Verlust durch Regulierung entsteht. Regulierung ist hier bei eine weg Overfitting zu vermeiden was das Phänomen beschreibt, wenn KI-Modelle sich 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. Ihn unserem Beispiel könnte es dann vorkommen, dass es die Ente vor einem Hintergrund der nicht im den Trainings Daten vorkommt nicht erkennt.

Darunter befindet sich die Learning Rate sie ist eine Variable in der Verlust funktion

Und ganz Unten wie viele schritte mein Computer pro Sekunde schafft.

### 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 der Linke zu dem GitHub Projekt ist der erste Anhang.

## Model mit Quantisierung optimieren

Man kann nun nach dem Training noch eine Quantisierung durchführen. Bei Post-Training quantisieren wird ein bereits fertig trainiertes Modell quantisiert, die Variablen der Gewichte des Modells werden von einem genaueren größeren Daten typen in einem kleineren ungenaueren umgewandelt, zum Beispiel von uint16 zu uint8. Das verringert die Model Größe und verbessert dessen Geschwindigkeit es verliert dabei aber auch an Präzision (tensorflow, 2022). Ich habe mein Model auch quantifiziert da ich möglichst hohe Performance rausholen wollte und um herauszufinden wie sich das auf das Modell auswirkt. Damit hatte ich dann auch 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 Ausprobiert ihre Performance gemessen und 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 gibt es zwei wichtige Maßeinheiten mit der man Objekt-Detektion 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 Leistung von Objekt-Detektion Modellen verglichen wird.

### Die Geschwindigkeit meines Modells

Um die Geschwindigkeit der Modelle zu messen hab ich mir von

Einer der Gründe warum mein Modell so langsam ist liegt daran das die Kamera die ich mit meinem Raspberry Pi verwende hatte die Logitech BRIO eine Auflösung von 4K hat. Dies ist mir aber erst später aufgefallen und es ist die einzige Webcam die wir hatten.

## Verbesserung des Modells durch neu Training

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 mehr Bildern trainiere. Habe ich noch 683 weitere Bilder gemacht und Gelabelt und mit diesen zwei weiteren Modellen trainiert, eins mit 650 Bildern Duck\_650 und eins mit 1000 Bildern Duck\_1000.

Um das Problem das es alle 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 teilweise funktioniert die späteren Modelle konnten deutlich besser zwischen der Ente und anderen gelben Objekten unterscheiden aber nicht immer.

[Bild gelbe Objekte besser]

Beim Training der Modelle mit mehr Bilder habe ich festgestellt das diese wie erwartbar deutlich länger im Training brauchten bis sie alle 40000 Steps abgeschlossen hatten das Modell mit 650 Bildern hat zwei und halb Stunden gebaucht und 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.

[Diagramm Geschwindigkeit]

Ihre mAP ist dafür aber deutlich Besser

[Diagramm mAP]

# Ergebnis

Zusammenfassend lässt sich sagen, dass

Das Pixy ist zwar deutlich schneller als das TFLite Model auf dem Raspberry dies liegt aber nicht daran, dass das Pixy besser Hardware hat, sondern daran das es den deutlich einfacheren Algorithmus nutzt und die Auflösung seiner Kamera so viel schlechter ist

# Fazit

Objekt-Detektion lässt sich heute schon auf Microcontrollern umsetzen

# Literaturverzeichnis

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

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

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 Screenshot meines Arduino Codes 5](#_Toc187704030)

[Abbildung 2 Screenshot von Tensorboard nach Ende des Trainings von Duck\_317 11](#_Toc187704031)

# KI-Verzeichnis

# Anhang

# 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.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ort, Datum Unterschrift