Unser Abschlussprojekt im Modul **Grundlagen der künstlichen Intelligenz**.

Dies ist eine Abschrift des [Wikis des Git-Repositorys](https://github.com/RFeuerhack/GKI_Abschlussprojekt/wiki)

**Dozent:** Prof. Dr. Frank Kammer

**Projektbeteiligte:** Philipp Händler, Robert Feuerhack, Felix Brübach

**Inhaltsverzeichnis**

**Tutorials**

* App installieren
* App benutzen
* Blender

**Aufgaben**

* 1: Objekt-Erkennung
* 2: Lösung nachvollziehen
* 3: Muster erstellen
* 4: Objekterkennung
* 5: 3D-Modell

**Bonus**

* 8: Rainbow Ball lösen

**Tutorials**

**App**

**Blender**

**1.** [Blender installieren](https://www.blender.org/download/)

**2.** Repository klonen

**3.** In Blender die jeweiligen Dateien (Rubix.blend für den Zauberwürfel bzw. Rainball.blend für den Rainbowball) im Ordner "blender" öffnen

**4.** Im scripting-context (rechte Seite) den erstellten JSON-String in das "colordic" einfügen

**5.** Das Script mit "Run Script" oben rechts ausführen (Alt + P)

**Aufgaben**

**1: Objekt-Erkennung**

**Aufgabe 1 (Objekt-Erkennung).** Untersuchen Sie verschiedenste Möglichkeiten mithilfe von neuronalen Netzen, Objekte auf einem Bild zu erkennen. Eine ausführliche Anleitung hierzu finden Sie zum Beispiel unter [medium.com/coinmonks/tensorflow-object-detection-with-custom-objects-34a2710c6de5](https://medium.com/coinmonks/tensorflow-object-detection-with-custom-objects-34a2710c6de5).

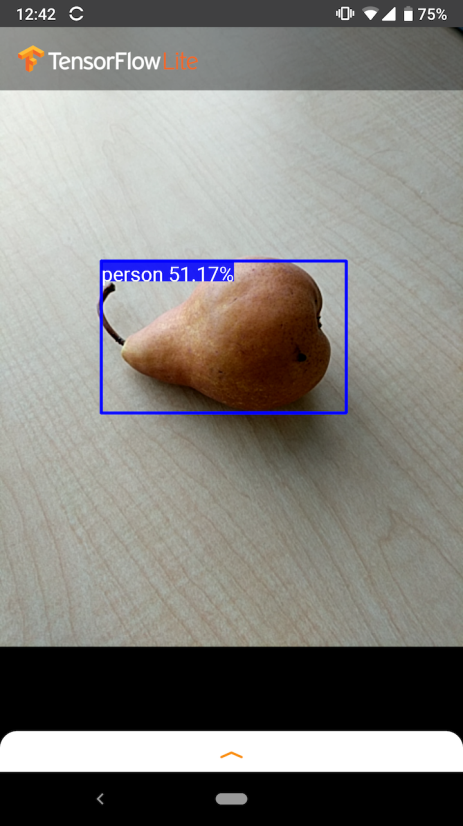
## Vorgehensweise

Wir haben nicht wirklich lange nach einer Alternative zu TensorFlow für diese Aufgabe gesucht, da es uns empfohlen wurde. Dies aber auch zu Recht, da in der Zeit, in der wir nach Alternativen gesucht haben nicht wirklich eine vergleichbares Framework gefunden haben, welches so mächtig und gut dokumentiert ist wie TensorFlow. Wir sind nicht dem in der Aufgabenstellung vermerkten Link gefolgt, sondern haben selber nach einer kurzen Google-Suche folgende Seite gefunden:   
  
<https://www.tensorflow.org/lite/models/object_detection/overview>   
  
Auf der Seite wird von Google sehr schön auf eine simple Art und Weise erklärt, was Object Detection ist. Außerdem wird das TFLite Framework vorgestellt, welches es ermöglicht TensorFlow-Modelle auch auf leistungsschwächeren Geräten, wie z.B. Smartphones und Raspberry Pis auszuführen. Auf dieser Seite kam uns erstmals die Idee eine App zu machen, um die folgenden Aufgaben zu lösen. Es hat für uns einfach Sinn gemacht ein Handy zu benutzen, da jedes Handy eine eingebaute Kamera hat und die Idee einen Videostream zu haben, in dem **Detections** markiert werden, mochten wir alle. Nach einer kurzen Recherche haben wir auf YouTube ein [Video](https://www.youtube.com/watch?v=3ORkKG_tVSg) gefunden, welches gezeigt hat, dass TFLite unsere Anforderungen erfüllen kann. Auf der Seite auf der Google TFLite vorstellt, haben sie auch ein Repository mit einem Beispielprojekt verlinkt, was uns sehr gelegen für [Aufgabe 2](https://github.com/RFeuerhack/GKI_Abschlussprojekt/wiki/Aufgabe-2:-L%C3%B6sung-nachvollziehen) kam.

**2: Lösung nachvollziehen**

**Aufgabe 2 (Lösung nachvollziehen).** Suchen und installieren Sie eine fertige Objekterkennungslösung, z.B. zur Erkennung von Werkzeugen. Testen Sie, wie gut und wie schnell diese fertige Lösung funktioniert. Hinweis: Zu bevorzugen sind Lösungen, die keine Cloud-Api benutzen.

## Vorgehensweise

Wie in [Aufgabe 1](https://github.com/RFeuerhack/GKI_Abschlussprojekt/wiki/Aufgabe-1:-Objekt-Erkennung) beschrieben, gibt es auf der Webseite, auf der wir uns zu Object Detection und TFLite informiert haben, einen Link zu einem Repository mit einem TFLite Beispielprojekt für Android, welches wir im Rahmen dieser Aufgabe versucht haben zum laufen zu bringen. Dies hat sich als sehr einfach herausgestellt. Wir mussten lediglich das Repository klonen und das verlinkte Verzeichnis in Android Studio als Projekt ausführen. Dann nur noch die App bauen und ausführen und schon hatten wir eine App auf unseren Handys mit der wir diverse Gegenstände (90 Stück z.B. Autos, Menschen, TV, Flasche, ...) erkennen konnten. Hier kam uns der Gedanke, dass wir theoretisch sobald wir selber Modelle haben diese nur im Beispielprojekt austauschen müssten, damit die vorgebaute App Zauberwürfel und Magic Rainbow Balls erkennen kann. Dies haben wir später auch zum vorläufigen testen der Modelle, bevor wir eine eigene App hatten, gemacht. Die App erkennt die Dinge, die Sie erkennen soll sehr gut und sehr schnell in dem Video der Handykamera, jedoch gibt es auch eine Menge sogenannter **False Positive Detections**. Das heißt das Dinge als etwas erkannt werden was sie eigentlich nicht sind.

**3: Muster erstellen**

**Aufgabe 3 (Muster erstellen).** Erstellen Sie viele Bilder von einem Zauberwürfel und einem Rainbow Ball Cube und markieren Sie diese beiden Objekte auf den Bildern mit einem passenden Label.

## Einleitung

In unserem Projekt haben wir uns für drei verschiedene Modelle für die Object-Detection entschieden, dabei handelt es sich um ein je ein Neuronales Netz, welches mithilfe von Trainingsdaten verschiedene Objekte erkennen soll. Bei der Erstellung der Trainingsdaten ist am wichtigsten zu beachten, dass es möglichst eindeutige Bilder sind, die sich von den anderen Trainingsdaten unterscheiden, um das Modell auf die gesuchten Objekte zu trainieren und nicht deren Umfeld. Zum Erstellen der Trainingsdaten wurde hier die Handykamera des iPhone 8 benutzt, danach wurden die gesammelten Bilder einzeln mithilfe der Python Anwendung „labelImg“ mit Labeln versehen.

## 1. Modell: Erkennung von Rubik's Cube und Rainbow Ball

Das erste Modell, dass wir trainiert haben sollte den Rubiks’s Cube und den Rainbow Ball erkennen. Hierfür wurde bei der Datenerstellung darauf geachtet, dass an möglichst verschiedenen Orten Bilder gemacht wurden, um kein Umfeld des Objekts dem Netz anzutrainieren, des Weiteren wurde darauf geachtet, verschiedene Formen der Objekte zu wählen (den Rubik’s Cube immer wieder neu verdrehen und die Anordnung der Bälle im Rainbow Ball durchmischen) , um auch hier keine Gleichmäßigkeit anzutrainieren. Der endgültige Trainingsdatensatz war mit 73 gelabelten Bildern zwar recht klein, hat aber bereits gute Resultate erzielen können, so hat die Object-Detection die beiden gesuchten Objekte zu rund 90 % immer erkannt, jedoch gab es ein Problem, dass das Modell aufwies einige andere Objekte die nicht erkannt werden sollten, wurden als eins der gesuchten Objekte erkannt.

## 2. Modell: Seitenerkennung des Rubik's Cube

Beim zweiten Modell ging es darum die Seiten des Rubik’s Cube zu erkennen, dafür hatten wir verschiedene Ansätze, hierbei hätte man für jede Farbe ein Label erstellen können und anhand der erkannten Felder die Anordnung auslesen. Wir haben uns für einen anderen Ansatz entschieden, bei dem wir die ganze Seite als Label wählen und die erkannte Seite dann mithilfe eines Algorithmus die Farben auslesen können. Bei den Testdaten für dieses Modell wurde hauptsächlich auf die Vielfältigkeit der Anordnung des Rubik’s Cubes geachtet. Das wichtigste bei diesem Modell war die Ausrichtung des Cubes zur Kamera, da die Labels immer genau die Seite des Cubes markiert, damit der Algorithmus die Farben in festen Abständen ermitteln kann. Mit einem Datensatz von 122 Bildern haben wir ein zufriedenstellendes Ergebnis erreichen können, wobei die Seiten des Cubes so erkannt werden konnten, um die Farben zu ermitteln.

## 3. Modell: Erkennung der Anordnung von Ringen und Bällen auf dem Rainbow Ball

Beim dritten Modell ging es, darum die verschiedenen Ringfarben und Ballfarben des Rainbow Balls zu erkennen, hierbei hatten wir zunächst den Ansatz die Bilder schwarz-weiß zu färben und dann die Struktur aus Ring und Ball zu erkennen. Jedoch muss man im Nachhinein die Farben erkennen, daher haben wir uns überlegt diesen Schritt zu vermeiden und statt ein Label für Ring und Ball zu nutzen, haben wir 24 Label genutzt für alle Ringe und Bälle der jeweiligen Farben. Beim ersten Training des Modells, mit 110 Bildern, gab es bereits gute Resultate, jedoch hat sich ein Problem mit den Farben Dunkelgrün und Cyan feststellen lassen. Aufgrund dieses Verhaltens haben wir uns dafür entschieden, den Datensatz zu erweitern, mit Fokus auf die Problemfarben. Nachdem wir den Datensatz um 100 Bilder erweitert haben, wurden die Ergebnisse besser, jedoch konnte man immer noch nicht die Farben Dunkelgrün und Cyan auseinanderhalten.

**4: Objekterkennung**

**Aufgabe 4 (Objekterkennung).** Lernen Sie die KI aus Aufgabe 2 so an, dass diese den Rainbow Ball Cube sowie einen Zauberwürfel erkennen. Nutzen Sie ein git-Repository für Ihre Softwareentwicklung.

## Einleitung

Wir haben uns bereits in [Aufgabe 1](https://github.com/RFeuerhack/GKI_Abschlussprojekt/wiki/Aufgabe-1:-Objekt-Erkennung) dazu entschieden TensorFlow bzw. TFLite zu benutzen. Nach kurzer Recherche haben wir herausgefunden, dass wir uns nun um diese Aufgabe mit den Technologien für die wir uns entschieden haben, mit der [TensorFlow Object Detection API](https://github.com/tensorflow/models/tree/master/research/object_detection) auseinandersetzen müssen.

## Vorgehensweise

Wir mussten herausfinden, wie man mit der [TensorFlow Object Detection API](https://github.com/tensorflow/models/tree/master/research/object_detection) eigene Modelle trainiert. Wir haben dieses [Turorial](https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi) und dieses [Tutorial](https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10) gefunden, welche sich als sehr hilfreich herausgestellt haben. Dort ist genau beschrieben, wie man die API zum laufen bringt und aus den in [Aufgabe 3](https://github.com/RFeuerhack/GKI_Abschlussprojekt/wiki/Aufgabe-3:-Muster-erstellen) entstandenen Trainingsdaten sogenannte TFRecords erstellen kann, mit denen man dann wiederum Modelle aus dem [Detection Model Zoo](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md) trainieren kann. Wir haben uns für das Modell mit dem Namen **ssd\_mobilenet\_v2\_coco** entschieden, da es nicht nur eine sehr gute Bewertung hat, sondern auch, weil wir nur Modelle, die auf **SSD** basieren nutzen können, da das TFLite Plugin für **Flutter** (Framework für die App) nur **YOLO** und **SSD** Modelle unterstützt und im [Detection Model Zoo](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md) keine Modelle, die auf **YOLO** basieren, angeboten werden. Wir haben eine Woche vor der Präsentation jede Nacht einen Rechner laufen lassen um Modelle für die verschiedenen Aufgabenstellungen zu trainieren. (~12h, 16GB Ram, 1080TI, i7-8770k) Diese Modelle wurden dann mithilfe der obigen Tutorials noch in das **.tflite** Format konvertiert und in das Beispielprojekt aus [Aufgabe 2](https://github.com/RFeuerhack/GKI_Abschlussprojekt/wiki/Aufgabe-2:-L%C3%B6sung-nachvollziehen) eingesetzt.

## Evaluierung

Die App hat nachdem wir das gewonnene Modelle zum Erkennen von Zauberwürfeln und Magic Rainbow Balls eingesetzt haben, alles diese sehr gut erkannt, jedoch gab es immernoch die in [Aufgabe 2](https://github.com/RFeuerhack/GKI_Abschlussprojekt/wiki/Aufgabe-2:-L%C3%B6sung-nachvollziehen) beschriebenen **False Positive Detections**.

# Aufgabe 5: 3D Modell

**Aufgabe 5 (3D-Modell).** Erweitern Sie die vorherige Aufgabe so, das aus mehreren Bildern eines Rainbow Balls/Zauberwürfels ein 3D-Modell berechnet und abgespeichert wird. Stellen Sie das 3D-Modell passend dar. Hinweis: Sie brauchen Bilder, wo mehrere Seiten eines Objektes gleichzeitig zu sehen sind.

## Einleitung

Bei der Aufgabenstellung ein 3D-Modell zu erstellen waren wir uns direkt einig dass wir in diesem Projektteil ein möglichst schönes Ergebnis erzielen wollen. Um dies umzusetzen haben wir uns verschiedene Möglichkeiten angeschaut und ausprobiert. Zuerst müssen wir uns aber überlegen, wie wir aus den über die Handykamera aufgenommen Bilder des Videostreams Informationen über den Zustand des Zaubewürfels, sowie des Magic Rainbow Balls gewinnen.

## Erklärung der Ideen

### Ermitteln der Zustände

#### Zauberwürfel

Unsere Strategie um den Gesamtzustand des Würfels herauszufinden, war folgende. Jede Seite wird einzeln gescannt. Wir müssen also die Objekterkennung so gestalten, dass sie Würfelseiten erkennt. Dann kann der Nutzer den Würfel vor die Kamera halten möglichst so, dass eine einzelne Seite erkennbar ist und die App erhält somit durch die Objekterkennung den Bildauschnitt mit der Würfelseite. Jetzt können wir die Farben der Würfelteile erhalten, indem wir vorerst die RGB an folgenden Koordinaten auslesen:   
  
(x, y, w (width), h (height) beschreiben ein Rechteck/den Bildausschnitt wo die Objekterkennung eine Würfelseite erkennt)   
  
( x + 1/6 \* w | y + 1/6 \* h) -> oben links   
  
( x + 3/6 \* w | y + 1/6 \* h) -> oben mitte   
  
( x + 5/6 \* w | y + 1/6 \* h) -> oben rechts   
  
( x + 1/6 \* w | y + 3/6 \* h) -> mitte links   
  
( x + 3/6 \* w | y + 3/6 \* h) -> mitte mitte   
  
( x + 5/6 \* w | y + 3/6 \* h) -> mitte rechts   
  
( x + 1/6 \* w | y + 5/6 \* h) -> unten links   
  
( x + 3/6 \* w | y + 5/6 \* h) -> unten mitte   
  
( x + 5/6 \* w | y + 5/6 \* h) -> unten rechts

Jeder ermittelte RGB-Wert wird in den Cielab ... todo

#### Magic Rainbow Ball

### Visualisierung

#### Erste Ansätze mit pygame

Bei der 3D-Modellierung wurden wir vor eine Herausforderung gestellt. Wie können wir die aus der Objekterkennung geparsten Daten als 3D-Modell realisieren. Zunächst kam uns das pygame package in den Sinn, mit dem man ein Koordinatensystem erstellen und Punkte darin zu Linien und Flächen verbinden kann. Diesen Ansatz verfolgten wir auch zunächst, bis wir mit den vorgefertigten geometrischen Körpern (Würfel und Dodekaeder) Probleme bekamen diese zu den eigentlich gewollten Zauberwürfel und Rainbow Ball zu modellieren. Durch die eher „steife“ Implementierung der Koordinatenpunkte und deren Verbindungen war es fast unmöglich die Kugeln für den Rainbow Ball zu erstellen, geschweige denn die Einfärbung ordentlich zu programmieren. Nach mehreren gescheiterten Versuchen sind wir zu dem Entschluss gekommen die pygame Variante zu verwerfen und uns lieber auf die umfangreichere Blender-Software zu konzentrieren, auf welche wir bei Recherchen gestoßen sind.

#### Die Blender Software

In Blender konnten wir für die Modellierung der geometrischen Körper die gleichen mathematischen Formeln und Koordinaten wie bei der pygame Variante nutzen, was uns einige anfängliche Arbeit ersparte. Ebenfalls vereinfachte die Blender Software durch schon vorgefertigte Formen wie Kugeln und Ringen die weitere Arbeit an den Modellen. Des weiteren war es von Vorteil, dass Blender die sogenannte Blender-Python-API besitzt. Mit Hilfe dieser war es uns dann möglich die Einzelnen Objekte mit einem Pythonscript zu verändern, verschieben und zu färben.

## Evaluierung

Die anfängliche Nutzung von pygame hat einige Zeit gekostet und war somit ein kleinerer Rückschlag. Dennoch konnten die mathematischen Koordinaten für den Dodekaeder übernommen werden.  
Die Nutzung von Blender gestaltete sich zunächt als schwierig, da wir noch nie mit dieser Software gearbeitet haben und zugehörige Dokumentation nicht sehr Einsteigerfreundlich ist. Dennoch denken wir dass dies der richtige Schritt war um das bestmögliche Ergebnis der 3D-Modellierung zu erzielen, denn sobald man sich in Blender eingearbeitet hat lässt sich sehr effizient mit der Software arbeiten.

## Fazit

Abschließend waren wir sehr zufrieden mit den Ergebnissen unserer Arbeit. Die anfänglichen Schwierigkeiten das 3D-Modell mit hilfe vom python package "pygame" zu realisieren als auch die ersten Hürden Blender zu nutzen konnten relativ schnell überwunden werden und haben uns nicht zu sehr aufgehalten. Die mathematischen Voraussetzungen für die modellierung der geometrischen Körper waren auch eher niedrig, weswegen wir mit ausschließlich guten Gefühlen diese Aufgabe abschließen konnten.

**Bonus**

# Aufgabe 8: Rainbow Ball lösen

**Aufgabe 8 (Rainbow Ball Cube lösen).** Entwickeln Sie eine KI, die es erlaubt, den Rainbow Ball Cube zu lösen.

## Einleitung

Wir haben uns für diese Aufgabe entschieden, da wir sie spannendste fanden. Ursprünglich dachten wir es wäre sehr schwer diese Aufgabe zu lösen, aber am Ende war es doch machbar.

## Erklärung der Ideen

Der **Magic Rainbow Ball** ist einem Schiebepuzzle sehr ähnlich, deshalb haben wir uns nochmal die Folien aus Vorlesung zu dem Thema angeschaut. Daher wussten wir, dass wir lediglich eine A\* Suche auf einen **Game Tree** durchführen müssen. Zuerst musste man jedoch eine Klasse schreiben, welche den Zustand eines **Magic Rainbow Balls** abbildet und Methoden wie vor allem **possibleMoves()**, **move()** und **manhattenDistance()** unterstützt. Als dies geschafft war, fiel es uns einfach den A\* Algorithmus, wie er im Skript beschrieben ist mit einer **Priority queue** zu implementieren.

## Evaluierung

Es gab nirgends wirklich bemerkenswerte Schwierigkeiten. Wenn man gut programmieren kann und die im oberen Punkt beschriebenen Aspekte versteht, ist es sehr leicht die Aufgabe zu lösen.

## Fazit

Unsere Implementierung findet in wenigen Sekunden einen Lösungsweg der immer ungefähr 20 Schritte benötigt um den **Magic Rainbow Ball** zu lösen, wenn dieser gut durchgemischt ist. Falls er nicht durchgemischt ist, werden natürlich kürzere Wege gefunden.