Prototypische Implementierung eines Straßenschilderkennungssystems für mobile Endgeräte

B.Sc. Matthias Haselmaier
Informatik und Mikrosystemtechnik
Hochschule Kaiserslautern
Zweibrücken, Deutschland
matthias.haselmaier@hs-kl.de

B.Sc. Andreas Brunnet
Informatik und Mikrosystemtechnik
Hochschule Kaiserslautern
Zweibrücken, Deutschland
andreas.brunnet@hs-kl.de

***Abstract*—Komplexere Fahrassistenzsysteme wie Straßenschilderkennung finden zunehmend Einzug in heutige Fahrzeuge. Eine flächendeckende Präsenz solcher Systeme ist im Straßenverkehr nicht gegeben. Ein Nachrüsten moderner Fahrassistenzsysteme ist für viele Fahrzeughalter keine Option. In dieser Ausarbeitung wird die Implementierung einer prototypischen Applikation für mobile Endgeräte zur Straßenschilderkennung beschrieben, welche eine Alternative zu herkömmlichen Nachrüstungssystemen darstellt. Mittels der TensorFlow-Object-Detection-API wurde ein Single-Shot-MultiBox-Detector (SSD) trainiert. Dabei handelt es sich um eine spezielle Convolutional-Neural-Network-Architektur. Als Traingsdatensatz wurde das German-Traffic-Sign-Detection-Benchmark-Datenset eingesetzt, in welchem Trainingsdaten für insgesamt 43 unterschiedliche Straßenschilder vorhanden sind. Die Ausführung der Inferenz erfolgt hierbei lokal auf dem mobilen Endgerät und benötigt keine entsprechende Internetverbindung. Auf einem Google Pixel (2016) können unter Verwendung dieses Netzwerks durchschnittlich 3 Bilder pro Sekunde analysiert werden.**

Keywords—Convolutional-Neural-Network, Single-Shot-MultiBox-Detector, Fahrassistenzsystem, Objekterkennung, TensorfFlow, Machine Learning, Straßenschilderkennung

I. Einleitung

Durch das Vorantreiben des autonomen Fahrens investieren große Autofirmen enorme Mengen an Ressourcen in sogenannte Autonomous-Vehicle-Driving-Systems (ADAS). Kommerzielle ADAS-Systeme in aktuellen Fahrzeugen enthalten unter anderem Traffic-Sign-Recognition-Systeme (TSR). Ältere Fahrzeuge können teilweise nachgerüstet werden. Dies ist mit hohen Kosten verbunden und kommt daher für viele Fahrer nicht infrage. In diesem Paper wird eine kostengünstige, prototypische Smartphone Applikation vorgestellt. Sie ist in der Lage, Straßenschilder in Echtzeit zu erkennen und kann den Fahrer vor dem Überschreiten der Höchstgeschwindigkeit warnen.

Im Folgenden werden zuerst die verwendeten Daten und Technologien beschrieben. Anschließend wird die Implementierung der Smartphone Applikation anhand von UML-Diagrammen dargestellt. Abschließend wird das resultierende System kurz bewertet sowie mögliche nächste Schritte und weitere Alternativen zu herkömmlichen Straßenschilderkennungssystemen vorgestellt.

II. Verwendete Daten und Technologien

In den folgenden Abschnitten werden die verwendeten Daten und Technologien beschrieben.

A. Datensatz

Das verwendete Netzwerk wurde mit dem German-Traffic-Sign-Detection-Benchmark-Dataset (GTSDB) des Instituts für Neuroinformatik [1] trainiert. Inhalt sind Aufnahmen deutscher Straßen mit entsprechenden Straßenschildern. Hierbei deckt der Datensatz 43 Schilder ab, die auf insgesamt 600 Trainingsbildern und 300 Evaluationsbildern abgebildet sind. Die Aufnahmen unterscheiden sich hinsichtlich der Lichtverhältnisse, des Sichtwinkels und der Entfernung zu den Schildern. Zusätzlich sind die Trainingsbilder annotiert. Das heißt, es wird angegeben, um welche Art von Schild es sich handelt und an welcher Position es sich befindet.

B. TensorFlow-Object-Detection-API

TensorFlow ist eine Open-Source-Softwarebibliothek für die numerische Berechnung mit Datenfluss-Graphen [2]. Dabei repräsentieren die Knoten des Graphen die mathematischen Operationen während die Kanten des Graphen die mehrdimensionalen Daten-Arrays darstellen, die sich zwischen diesen bewegen. TensorFlow bietet eine Python-API zum Trainieren von künstlichen neuronalen Netzwerken an. Weiterhin werden Bibliotheken bereitgestellt, um trainierte Netzwerk unter weiteren Programmiersprachen wie C++ oder Java zu laden und nutzen zu können.

Aufbauend auf der TensorFlow-API ermöglicht die TensorFlow-Object-Detection-API (TFOD-API) [3] eigene Modelle zur Detektion von Objekten zu trainieren. Das Detektieren von Objekten umfasst hierbei das Klassifizieren der sich in einem Bild befindlichen Objekte und die Lokalisierung dieser Objekte. Es werden bereits Implementierungen verschiedener Netzwerkarchitekturen bereitgestellt, welche auch als vortrainierte Varianten verfügbar sind. Werden der TFOD-API ein vortrainiertes Netzwerk und ein Datenset bereitgestellt, trainiert sie das Netzwerk auf die neuen Daten. Netzwerke, die durch die TFOD-API trainiert wurden, geben erkannte Objekte in vier Variablen aus. Die erste entspricht der Anzahl der erkannten Objekte im Bild. In den letzten 3 Variablen wird die Art des erkannten Objekts, die Position der BoundingBox und das Konfidenzniveau, das angibt wie sicher das jeweilige Objekt erkannt wurde, definiert.

C. Netzwerk

Beim verwendeten künstlichen neuronalen Netzwerk handelt es sich um eine Kombination der SSD Metaarchitektur und der MobileNet Architektur. Dieses Netzwerk ist ein reines Convolutional-Neural-Network. Diese Architektur wurde gewählt, da sie eine schnelle Analyse der Bilder erlaubt und

dennoch eine gute Genauigkeit erreicht. Im Folgenden werden die Architektur und das Training kurz beschrieben.

1) MobileNet

Basis des Netzwerks bildet die MobileNet [5] Architektur. Entwickelt wurde diese aufgrund der Anforderungen an Latenz, Größe und Klassifizierungspräzision, die im Bereich der Embedded Systeme, also der Robotik, dem autonomen Fahren oder auch auf mobilen Endgeräten gelten. Kern der Architektur bilden tiefenseparierbare Filter. So werden Standard Convolutions in tiefenweise Convolutions faktorisiert. Daraus resultieren 1x1 Punkt-Convolutions. Hiermit wird sowohl die Rechenzeit als auch die Größe des Netzwerks reduziert. Abschluss dieser Architektur bildet eine Schicht, die ausgehend aus den resultierenden Punkt-Convolutions eine Linearkombination bildet, um neue Features zu generieren.

2) SSD

Auf Basis des MobileNet ist die Single-ShotMultiBox-Detector Metaarchitektur [4] implementiert. Das SSD ist ein einzelnes Deep-Neural-Network zur Objekterkennung in Bilddaten. Als Output werden Bounding-Boxes und korrespondierende Konfidenzniveaus zu erkannten Objekten geliefert. Zur Generierung der Bounding-Boxes greift die Architektur auf eine Menge an vordefinierten Boxen in verschiedenen vordefinierten Seitenverhältnissen zurück, aus denen für jedes erkannte Objekt die passgenauste Box ausgewählt und sofern nötig in ihrer Größe korrigiert wird.

3) Training

Es wurde ein vortrainiertes SSD/MobileNet-Netzwerk aus dem TensorFlow-Detection-Model-Zoo verwendet. Zum Vortraining des Netzes wurde das Common-Objects-in-Context (COCO) [6] Datenset verwendet, das über 330.000 Bilder, eingeteilt in 80 Objektkategorien verfügt. Darauf aufbauend wurde das Netz unter Verwendung des GTSDB-Datensets auf das Erkennen von Straßenschildern für 20.000 Schritte weiter trainiert.

III. Implementierung

Zur Implementierung eines Systems zur Erkennung von Straßenschildern wurde die Applikation in 4 Klassen aufgeteilt. In Fig. 1 ist abgebildet, wie die Klassen zusammenhängen. Durch *TobiNetwork* wird das mit TensorFlow trainierte CNN abstrahiert. Erkannte Straßenschilder werden der restlichen Anwendung über Objekte der Klasse *DetectedObject* bereitgestellt. *BoundingBoxView* zeigt den Kamerafeed des Smartphones und visualisiert erkannte Straßenschilder. Die Geschwindigkeit des Fahrzeugs wird in der Klasse *GpsHandler* berechnet und über die *SpeedChangeListener*-Schnittstelle an die Klasse *MainActivity* übergeben. *MainActivity* implementiert die üblichen Android-Activity-Lifecycle-Methoden und stellt das Bindeglied zwischen den anderen Klassen dar. Im Folgenden wird auf die Implementierung der einzelnen Klassen genauer eingegangen.

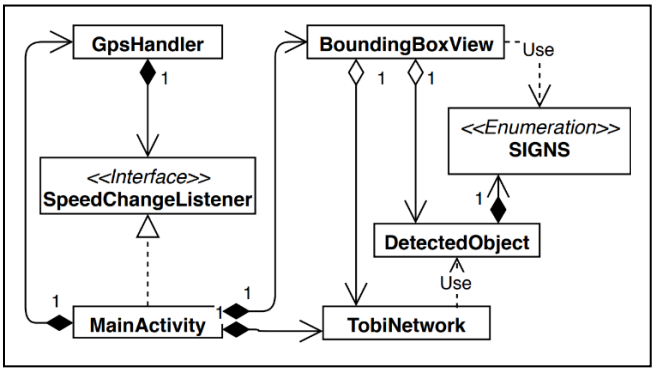

Fig. 1. Implementierte Klassen

A. MainActivity

Die *MainActivity*-Klasse stellt die Hauptklasse der Applikation dar. Sie erbt von *AppCompatActivity* aus der Android-API und implementiert die notwendigen Android-Activity-Lifecycle-Methoden. Wie in Fig. 1 zu sehen ist, besitzt sie je eine Referenz auf eine Instanz der anderen 3 Klassen. Wird die Applikation gestartet, werden die Instanzen der anderen Klassen initialisiert. Die benötigten Systemressourcen werden belegt, beziehungsweise freigegeben, wenn die Applikation in den Vordergrund gerufen wird oder in den Hintergrund verlagert wird. *MainActivity* ist auch dafür zuständig, die benötigten Berechtigungen vom Benutzer anzufordern, um auf die Kamera und die Position des Smartphones zuzugreifen. Weiterhin implementiert sie das *SpeedChangedListener*-Interface, um die aktuelle Geschwindigkeit zu erhalten. Wird die aktuelle Höchstgeschwindigkeit, angegeben durch das zuletzt erkannte Höchstgeschwindigkeitsschild, überschritten, spielt *MainActivity* einen Warnton ab.

Neben diesen Funktionalitäten stellt *MainActivity* UI-Elemente zur Verfügung, um das minimale Konfidenzniveau für erkannte Straßenschilder zu setzen und die Geschwindigkeitsüberwachung ein- und auszuschalten.

B. GpsHandler

Durch die *GpsHandler*-Klasse wird die Android-Location-API gekapselt. Wie in Fig. 2 gezeigt wird, implementiert die Klasse das *LocationListener* Interface der Android-API. Wird die *start()*-Methode aufgerufen, registriert sich das *GpsHandler*-Objekt beim *LocationManager*, um über Positionsänderungen des Smartphones informiert zu werden. Um sich beim *LocationManager* wieder abzumelden, wird die *stop()*-Methode aufgerufen. Wurde vom *LocationManager* eine neue Position ermittelt, wird die *onLocationChanged()*-Methode aufgerufen. In ihr wird die Geschwindigkeit des Smartphones berechnet, indem die vorherige und aktuelle Position sowie die verstrichene Zeit verglichen werden. Diese Geschwindigkeit wird anschließend zusammen mit der neuen Position an die *onSpeedChanged()*-Methode des *SpeedChangeListeners* übergeben.

C. TobiNetwork

TensorFlow stellt eine Bibliothek für Android-Systeme zur Verfügung, mit welcher es ermöglicht wird, bereits trainierte Modelle auszuführen. *TobiNetwork* wird in Fig. 3 gezeigt und kapselt diese Bibliothek. Das Interface zur Klasse ist sehr schlicht gehalten. Die wichtigste Methode ist *predict()*, durch welche die Erkennung der Straßenschilder gestartet wird.

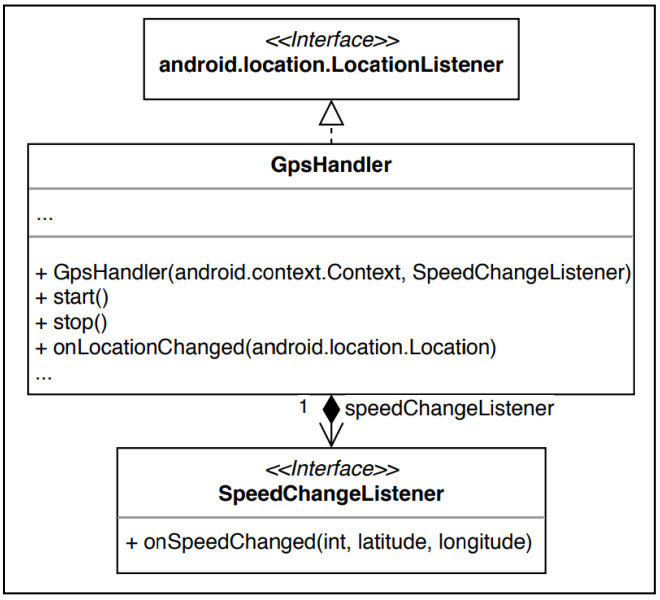

Fig. 2. Ausschnitt der GpsHandler-Klasse

Sie nimmt ein Byte-Array sowie die Bildbreite und -höhe entgegen und gibt ein Array von *DetectedObject*s zurück. Dieser Datentyp wurde eingeführt, da das trainierte neuronale Netzwerk die erkannten Objekte durch 4 Gleitkommawerte für die Bounding-Box, einer Ganzzahl für die Art des Schildes und einem Gleitkommawert für das Konfidenzniveau darstellt. Weiterhin bietet *TobiNetwork* Methoden an, um das minimale Konfidenzniveau zu setzen und zu lesen. Es werden nur erkannte Objekte von *predict()* zurückgegeben, deren Konfidenzniveau mindestens dem übergebenen Wert entspricht.

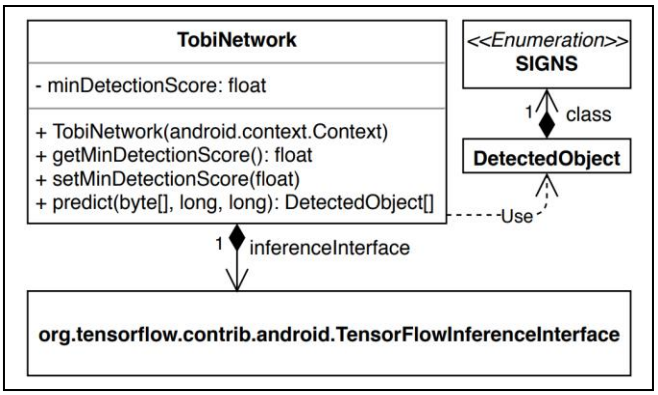

Fig. 3. Ausschnitt der TobiNetwork-Klasse

Für jedes Schild, auf das das Netzwerk trainiert wurde, ist in der Enumeration *SIGNS* ein Wert definiert. Weiterhin besitzt sie die statische Methode *mapSignToImageResource()*, welche das jeweilige Schild auf einen Ressourcen-Identifikator abbildet. Durch ihn können die entsprechenden Bilddaten geladen werden, um die Straßenschilder zu visualisieren.

D. BoundingBoxView

Die *BoundingBoxView*-Klasse ist dafür zuständig, den Kamerafeed sowie die erkannten Straßenschilder anzuzeigen. Sie konfiguriert die Hauptkamera des Smartphones so, dass auf einem *TextureView* das aktuell erfasste Bild angezeigt wird. Auf diesem *TextureView* registriert sie sich als *SurfaceTextureListener*, um benachrichtigt zu werden, wenn sich das angezeigte Bild ändert. Wurde von der Kamera ein neues Bild erfasst und dem *TextureView* zur Verfügung gestellt, wird die *onSurfaceTextureUpdate()* Methode aufgerufen. In dieser Methode wird das aktuelle Bild aus dem *TextureView* ausgelesen und zur Verarbeitung im *TobiNetwork* vorbereitet. Die erkannten Straßenschilder werden mit einer Bounding-Box als Overlay in einem zweiten *TextureView* hervorgehoben. Weiterhin werden erkannte Straßenschilder am unteren Rand des Bildschirms für 5 Sekunden angezeigt. Damit die Applikation ansprechbar bleibt, wird die Erkennung der Straßenschilder asynchron durchgeführt. Die Kamera des Smartphones erfasst Bilder schneller als sie analysiert werden können. Da zu jedem Zeitpunkt immer nur ein Bild von der TensorFlow Bibliothek analysiert werden kann, wird ein *AtomicBoolean* in *onSurfaceTextureUpdate()* verwendet. Nur wenn dieser Boolean False ist, wird ein neuer Erkennungsprozess gestartet.

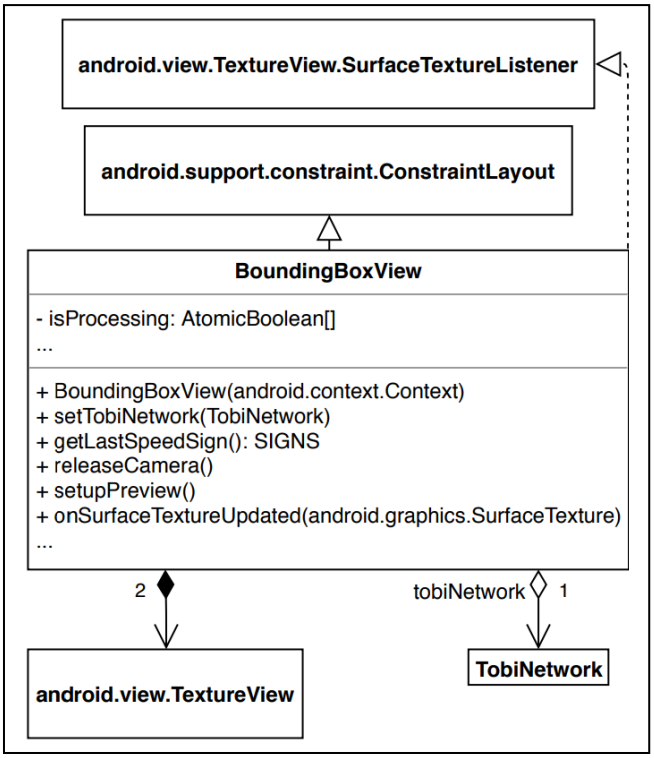

Fig. 4. Ausschnitt der BoundingBoxView-Klasse

Durch *getLastSpeedSign()* wird das zuletzt erkannte Straßenschild zurückgegeben. Diese Methode wird in *MainActivity* aufgerufen, um den Fahrer auf das Überschreiten der Höchstgeschwindigkeit aufmerksam zu machen.

Da durch das Verwenden der Kamera Systemressourcen genutzt werden, müssen diese wieder freigegeben werden, sobald die Applikation minimiert oder beendet wird. Hierzu wird die *releaseCamera()* Methode angeboten, welche die belegten Systemressourcen wieder frei gibt. Kehrt die Applikation aus dem Hintergrund zurück, kann die Kamera durch *setupPreview()* erneut Konfiguriert werden.

IV. Bewertung

Obwohl die gewählte Netzwerkarchitektur im Verhältnis zu anderen Architekturen ressourcenschonend ist, stellt das Erkennen von Straßenschildern hohe Anforderungen an das Smartphone. Gängige Smartphones verfügen nur über eine passive Kühlung. Dies kann bereits nach wenigen Minuten

dazu führen, dass das Analysieren der Bilder länger dauert, um ein Überhitzen des Smartphones zu verhindern. Da das Smartphone auf dem Armaturenbrett angebracht wird, ist es der direkten Sonneneinstrahlung ausgesetzt, wodurch dieser Effekt verstärkt wird.

Ein Nachteil der gewählten Netzwerkarchitektur ist, dass sie die zu analysierenden Bilder auf 300x300 Pixel skaliert. Dies ist ein Grund, weshalb diese Architektur schneller ist, als andere. Allerdings geht hierdurch auch Information verloren. Somit wird besonders das Erkennen von weit entfernten Schildern erschwert.

Auf einem Google Pixel (2016), welches mit der Snapdragon 821 CPU von Qualcomm und einer Adreno 530-GPU ausgestattet ist, können bis zu 3 Bilder pro Sekunde analysiert werden. Innerorts, bei einer Geschwindigkeit bis 50 km/h, werden Höchstgeschwindigkeitsschilder sowie Vorfahrtsschilder sicher erkannt. Blaue Straßenschilder, welche die vorgegebene Fahrtrichtung angeben, werden Aufgrund ihrer Ähnlichkeit häufig verwechselt. Auch auf Autobahnen bei einer Geschwindigkeit von 120 km/h werden Höchstgeschwindigkeitsschilder erkannt. Aufgrund der Geschwindigkeit des Fahrzeugs und der Dauer der Analyse, werden dem Fahrer die erkannten Schilder jedoch erst angezeigt, wenn sie bereits passiert wurden.

V. Ausblick

Durch die vorgestellte Applikation wurde gezeigt, dass durch Convolutional-Neural-Networks auf modernen Smartphones prinzipiell ein günstiges Straßenschild-erkennungssystem als Alternative zu herkömmlichen TSR-Systemen realisiert werden kann. Um das bestehende System zu verbessern, könnte der verwendete Datensatz um weitere Bilder und Szenarios erweitert werden. Aktuell befinden sich nur Bilder bei sonnigem Wetter im Datensatz. Damit das System Straßenschilder robuster erkennt, könnten demnach Bilder bei Nacht oder während der Dämmerung ergänzt werden. Zudem fehlen Trainingsbilder bei Regen, Nebel und Schneefall. Um das Erkennen von weit entfernten Straßenschildern zu verbessern, könnte ein adaptiver Zoom basierend auf der Geschwindigkeit implementiert werden. So können Schilder erkannt werden, bevor sie hinter dem Fahrzeug liegen.

Neben der Realisierung als Smartphone Applikation könnten neue Single-Board-Computer genutzt werden, welche dedizierte Hardware besitzen, um künstliche neuronale Netzwerke auszuführen. Zwei Beispiele hierfür sind das Coral Dev Board von Google oder das Jetson Nano Entwicklerkit von NVIDIA. Solche Single-Board-Computer haben genügend Leistung, um Netzwerkarchitekturen zu unterstützen, die eine höhere Genauigkeit erreichen, als das hier verwendete Netzwerk. Im Gegensatz zur Implementierung als Smartphone Applikation wird hierbei jedoch neue Hardware benötigt. Da die meisten Personen ein modernes Smartphone besitzen, muss bei der hier vorgestellten Lösung lediglich eine Applikation installiert werden.

References

[1] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, C. Igel, “Detection of Traffic Signs in Real-World Images: The {G}erman {T}raffic {S}ign {D}etection {B}enchmark”, The 2013 International Joint Conference on Neural Networks (IJCNN), Dallas, TX, 2013, pp. 1-8.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, et al., “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems”, arXiv:1603.04467 [cs], März 2016

[3] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, et al., “Speed/accuracy trade-offs for modern convolutional object detectors”, arXiv:1611.10012 [cs.CV], November 2016.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, et al., “SSD: Single Shot MultiBox Detector”, arXiv:1512.02325[cs.CV], Dezember 2015

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, arXiv:1704.04861 [cs], April 2017

[6] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, et al., “Microsoft COCO: Common Objects in Context”, arXiv:1405.0312[cs.CV], Mai 2014