

Medien Projekt

Gesichtserkennung

Gesichtserkennung mittels Neuronalen Netzen

Eingereicht am:

26 August 2019

Eingereicht von:

Jakob Lambert
Süldorfer Mühlenweg 31
22123 Hamburg
Tel.: +49 17632643656
E-mail: minf101027@fh-wedel.de

Referent:

Prof. Dr. Dennis Säring
Fachhochschule Wedel
Feldstraße 143
22880 Wedel
Phone: (041 03) 80 48-43
E-mail: sdg@fh-wedel.de

Inhaltsverzeichnis

Abbildungsverzeichnis	III
List of Listings	IV
1 Einleitung	1
2 Verwendete Tools und Bilddatensätze	2
2.1 Tool 1: Tensorbox	2
2.2 Tool 2: Object detection API	2
2.3 Verwendete Bilddatensätze	2
3 Herangehensweise	3
4 Bilderkennung mit dem Tool Tensorbox	4
4.1 Aufbau der Trainingsumgebung	4
4.1.1 Parameteranpassung beim Training	5
4.2 Trainingsergebnisse	5
4.3 Testphase	6
5 Bilderkennung mit der Object detection API	8
5.1 Aufbau der Trainingsumgebung	8
5.2 Training in der Cloud	9
5.3 Ergebnisse	9
5.4 Testphase	11
6 Die Applikation	13
6.1 Projektaufbau	13
6.2 Arbeiten mit dem Applikation	13
6.3 Wichtige Funktionen	13
7 Zusammenfassung und Ausblick	15
Literatur	16

Abbildungsverzeichnis

4.1	Verlauf des Regression Loss bei unterschiedlichen Trainingsläufen	5
5.1	Die beiden Abbildungen zeigen den Classification Loss und den Localization Loss des <i>rcnn_nas_coco</i> -Models	9
5.2	Der Total Loss für das “rcnn_nas_coco“-Model	10
5.3	Der Total Loss für das “ssd_mobilenet_coco“-Model	10
5.4	Die beiden Abbildungen zeigen den Classification Loss und den Localization Loss des “ssd_mobilenet_coco“-Models	10

List of Listings

1

Einleitung

Bei der automatischen Gesichtserkennung werden zwei Systeme betrachtet, die sich in ihrer Zielgebung unterscheiden. Das erste System hat das Ziel, Gesichter auf einem Bild zu lokalisieren und wird als Face Detection bezeichnet. Das zweite System baut auf dem ersten auf und hat das Ziel, Gesichtsmerkmale zu erkennen, um die Gesichter dann einer Person zuordnen zu können. Dieses System wird als Face Recognition bezeichnet.

Der größte Durchbruch in dem Bereich der Face Detection wurde durch die Viola-Jones-Methode erreicht. Bei dieser 2001 von Paul Viola und Michael Jones vorgeschlagenen Methode werden nicht klassische Algorithmen, sondern maschinelles Lernen eingesetzt, um Bildmuster zu erkennen. Das Verfahren erwies sich als sehr robust und schnell und ermöglichte es, Gesichter mit hoher Genauigkeit in Echtzeit zu erkennen, was für den Einsatz in mobilen Kameras wichtig ist.

Eine weiter populäre Lösung für die Mustererkennung ist die Verwendung von Neuronalen Netzen. Diese Lösung benötigt jedoch eine große Menge an Datensätzen. Im Rahmen des Medien-Projektes war es die Aufgabe, eine Applikation zu entwickeln, die in der Lage ist, alle Gesichter auf einem Bild zu erkennen, die erkannten Gesichter auszuschneiden und in einen Ordner zur Weiterverarbeitung abzulegen. Für die Gesichtserkennung sollte die Methodik der Neuronalen Netzen eingesetzt werden.

Zur Lösung der Aufgabe wurden in der Arbeit zwei mögliche open-source Tools untersucht, die grundlegende Objekterkennungsmodelle bereitstellen und damit das Trainieren der Neuronalen Netze ermöglichen. Die entwickelte Applikation kann wahlweise beide Tools benutzen.

Diese Arbeit stellt in Kap. 2 die verwendeten Tools und die für das Training verwendeten Bilddatensätze dar. Im Kap. 4 wird die Herangehensweise und die Trainingsumgebung für die Testläufe beschrieben, im Kap. 5 die mit den beiden Tools erzielte Gesichtserkennung. Kap. 6 beschreibt die entwickelte Applikation, in Kap. 7 werden die erzielten Ergebnisse bewertet und ein Ausblick auf die weitere Entwicklung gegeben. Eine Zusammenfassung und das Quellenverzeichnis schließen die Arbeit ab.

2

Verwendete Tools und Bilddatensätze

2.1 Tool 1: Tensorbox

Tensorbox ist eine Framework zum Trainieren eines Neuronalen Netzwerkes ([1](#), [2](#)), insbesondere im Hinblick auf das Erkennen von Objekten auf Bildern. Das Framework basiert auf Tensorflow, dies ist ein von Google entwickeltes Framework zur Verarbeitung von Datenströmen, das im Wesentlichen beim maschinellen Lernen eingesetzt wird und in Python und C++ entwickelt wurde. Zum Trainieren der Neuronen wird der GoogLeNet-OverFeat Algorithmus verwendet. Zusätzlich stellt Tensorbox ein Allgemeines Netzwerk zum Erkennen von Objekten zur Verfügung. Dieses Netzwerk kann von den Nutzern durch eigenes Training auf eine spezielle Objektkategorie angepasst werden. Zusätzlich kann das Training durch das Anpassen von bestimmten Parametern optimiert werden.

2.2 Tool 2: Object detection API

Die Object Detection API ist ein Open Source Framework ([3](#), [4](#)), welches wie auch Tensorbox mehrere Objekte auf einem Bild lokalisiert und identifiziert. Dieses Framework wurde direkt von Google entwickelt und basiert auch auf Tensorflow. Die API stellt vortrainierte Objekterkennungsmodellen zur Verfügung. Zusätzlich zu den vorgegebenen Modellen können auch eigene Modelle entwickelt werden. Das Training der Neuronalen Netze kann lokal oder mit der Cloud Machine Learning Engine von Google durchgeführt werden.

2.3 Verwendete Bilddatensätze

Für das Training des Neuronalen Netzes wird der WIDER Face Datensatz verwendet ([5](#)). Dieser beinhaltet 32.203 Bilder und 393.703 markierte Gesichter. Der WIDER Face Datensatz wird in 61 Klassen organisiert. Nach einem Zufallsverfahren wurden aus den einzelnen Klassen 40% der Bilder für das Training, 10% für die Validation und 50% für einen Anwendungstest verwendet.

3

Herangehensweise

Nach einer Einarbeitung in die Funktionsweise der Tools wurde die Arbeit in folgenden Schritten durchgeführt:

Schritte	Beschreibung
Prozessieren der Wider-Daten	Dieser Schritt war notwendig, um die verfügbaren Bildinformationen mit den Gesichtserkennungsinformationen zu verknüpfen.
Training	Mit den beiden Tools wurde das Training z.T. mit mehreren Parametereinstellungen durchgeführt.
Auswertung des Trainings	Die von den Tools aufgezeichneten Trainingsergebnisse wurden ausgewertet. Eine Validierung wurde nicht durchgeführt.
Test der App am Testdatensatz	An dem Testdatensatz wurde geprüft, wie hoch die mit der trainierten Gesichtserkennung erreichte Genauigkeit ist. Der Test wird nur an einer begrenzten Zahl von Bildern durchgeführt.

Tabelle 3.1: Herangehensweise

4

Bildererkennung mit dem Tool Tensorbox

4.1 Aufbau der Trainingsumgebung

In diesem Abschnitt wird der Aufbau der Trainingsumgebung für Tensorbox beschrieben, Details können https://github.com/DevJakobL/Face_detection entnommen werden.

Die Ordnerstruktur ist wie folgt aufgebaut:

```
Face_detection
├── TENSORBOX
│   └── hypes
├── data
│   ├── wider_face_split
│   ├── WIDER_test
│   ├── WIDER_train
│   └── WIDER_val
└── output
```

Der Ordner **TENSORBOX** beinhaltet Tensorbox und dessen Skripte zum Trainieren und Validieren von Neuronalen-Netze. In dem Unterordner **hypes** befinden sich Dateien, um das Training mit Tensorbox zu beeinflussen.

In dem Ordner **data** wird der Wider-Face Datensatz gespeichert. Dieser muss zunächst heruntergeladen und vor der Speicherung aufbereitet werden. Tensorbox benötigt die Daten als json-Datei, in der die Informationen über den Pfad der einzelnen Bilder sowie der auf den jeweiligen Bildern vorhandenen Gesichter mit ihren Koordinaten verbunden sind. Die Felder sind (6):

image_path gibt den Pfad zu dem Bild, welches auf Gesichter untersucht werden soll.

rects beinhaltet wiederum eine Liste mit den Koordinaten aller Gesichter auf dem Bild. Dabei dienen die Endpunkte der Hauptdiagonalen des Rechtecks zur Kennzeichnung der Lage eines Gesichts.

Um die Informationen zu verknüpfen wird das Skript *wider_parser.py* für alle drei Teilmengen der Bilder ausgeführt und damit die Dateien **wider_test.json**, **wider_train.json** und **wider_val.json** generiert. Dazu verwendet das Script die externe Library *python-widerface* (7), welches die Dateien aus **wider_face_split** Ordner aufruft, so dass sie von dem Script in der json-Datei gespeichert werden können.

In der obenstehenden Ordnerstruktur beinhalten die Ordner **WIDER_test**, **WIDER_train** und **WIDER_val** die jeweiligen Bild-Daten wie auch die json-Dateien.

In dem Ordner **Output** werden die durch das Training erstellten Daten gespeichert.

4.1.1 Parameteranpassung beim Training

Für das Training der einzelnen Modelle wurden unterschiedliche Einstellungen ausprobiert. Zum einen wurde das Residual Neural Networks (ResNet) dazu geschaltet, welches dem Neuronalen Netz ermöglicht überflüssige Layer zu überspringen. Außerdem wurden während der Arbeit drei unterschiedliche Lernraten getestet. Dieser in der Toolbox definierbare Parameter (learning rate) verändert die Änderungsgeschwindigkeit der Gewichte in Bezug auf den Verlustgradienten(8)[S. 16].

4.2 Trainingsergebnisse

Zur Bewertung der mit Tensorbox trainierten Modelle wird der Prüfparameter Regression Loss herangezogen. Dieser kennzeichnet die Genauigkeit, mit der es sich bei dem gefundenen Objekt um ein Gesicht handelt. Der Parameter ist gleichzusetzen mit dem Total Loss aus dem Abschnitt 5.3. Der Regression Loss wird durch das Tool Tensorbox kontinuierlich gespeichert und kann nachträglich ausgewertet werden.

In der Abbildung 4.1 ist der Verlauf der Regression Loss über Zahl der Bilder (X-Achse) dargestellt. Die einzelnen Linien unterscheiden sich in den Trainingseinstellungen.

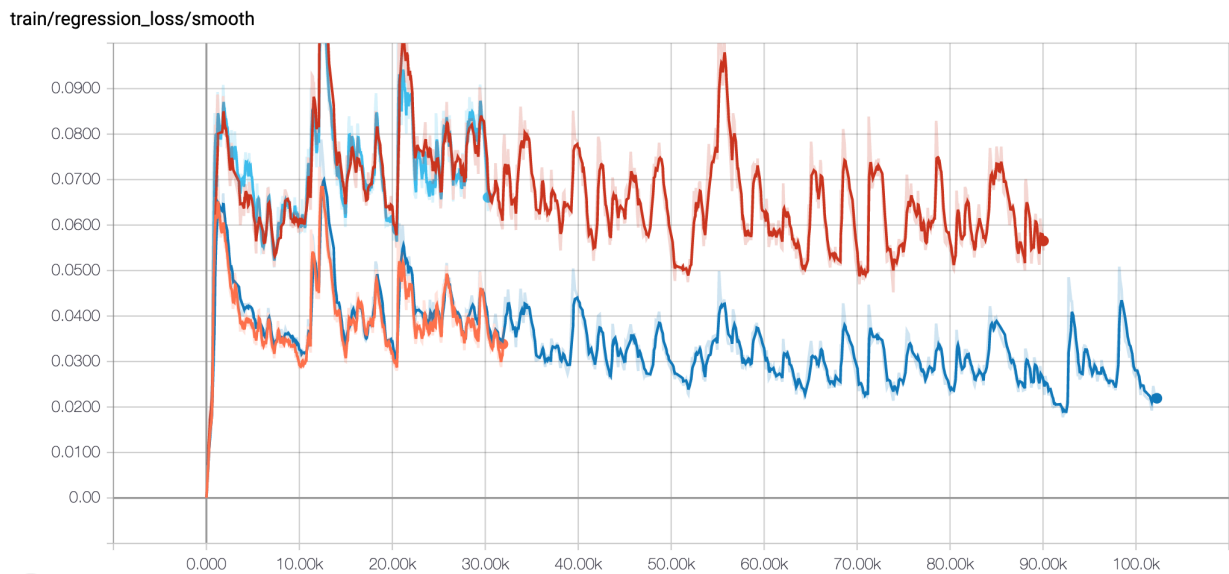


Abbildung 4.1: Verlauf des Regression Loss bei unterschiedlichen Trainingsläufen

- Die blaue Linie beschreibt das Training mit der voreingestellten Lernrate von 0,001.
- Die hellblaue Linie beschreibt das Training mit einer Lernrate von 0,00001,
- Die rote Linie beschreibt das Training mit einer Lernrate von 0,1.
- Die orange Linie beschreibt das Training mit voreingestellter Lernrate und eingeschaltetem Residual Neural Networks.

Anhand des Verlaufs des Regression Loss bei den vier getesteten Einstellungen wird deutlich,

- dass die Zuschaltung des Residual Neural Networks den Prüfparameter nur unwesentlich beeinflusst.

- dass die voreingestellte Lernrate (0,001) sich deutlich von den um den Faktor 100 erhöhten bzw. verminderten Lernraten unterscheidet.
- Dass mit den modifizierten Lernraten auch bis zum Ende des Trainingslaufs nicht die Lernerfolge erzielt werden können wie bei der Voreinstellung.

4.3 Testphase

Die Testphase wurde mit dem Standardmodell von Tensorbox durchgeführt. Dabei wurde an 5 Bildern aus dem WIDER_test Datensatz mit der in Kapitel 6 beschriebene Applikation geprüft, ob das trainierte Model Gesichter erkennen kann. Die Bilder sind im folgenden aufgeführt und enthalten 34 Gesichter wovon 10 Gesichter gefunden wurden, wobei davon doppelt gefundene Gesichter nicht beachtet werden.



Bild	Beschreibung
 <p>Download from Dreamstime.com 3035-4386 Dmitry Shironosov Dreamstime.com</p>	2/4
	3/4

Tabelle 4.1: Herangehensweise

4 Bilderkennung mit dem Tool Tensorbox




Bild	Beschreibung
	<p>1/18</p>
	<p>3/4</p>
	<p>1/4</p>

Tabelle 4.1: Herangehensweise

Zusammenfassend wird festgestellt, dass Gesichter gefunden wurden, die für den Menschen Schwer zu erkennen sind und im Gegensatz dazu werden Gesichter ignoriert die gut zu erkennen sind.

5

Bilderkennung mit der Object detection API

5.1 Aufbau der Trainingsumgebung

In diesem Abschnitt wird auf den Aufbau der Trainingsumgebung der Object Detection API eingegangen, Details können unter dem Link <https://github.com/DevJakobL/gCloudFacedetection> entnommen werden.

Der **Aufbau der Lokalen Struktur** wird im folgenden beschrieben:

```
ggCloudFacedetection
├── data
│   ├── WIDER_test
│   ├── WIDER_train
│   ├── WIDER_val
│   ├── wider_face_split
│   └── wider_to_tfrecord
├── dist
├── slim
├── object_detection
└── object_detection.egg-info
```

In dem Wurzelverzeichnis werden die Konfiguration Dateien abgelegt, die für die Definition des Trainings verwendet werden. Die Ordner **object_detection**, **object_detection.egg-info**, **slim** und **dist** stellt die Object Detection API zur Verfügung.

In dem Ordner **RESULTS** werden die in der Cloud gespeicherten Modelle lokal gesichert.

Der Ordner **data** beinhaltet den Wider Face Datensatz der aus dem Internet geladen werden muss (WIDER_test, WIDER_train und WIDER_val). Des weiteren beinhaltet der Ordner die benötigten TFRecords, die die Object detection API benötigt um die Bilddaten serielle verarbeiten zu können. Die Skripte, die für die Umwandlung der Bilddaten zuständig sind befinden sich in dem Ordner **wider_to_tfrecord**. Die Umwandlung der Datensätze basiert auf dem Projekt widerface-to-tfrecord (9) und folgt den Standards der offiziellen Tensorflow Object Detection API (10). Für das Training mit der API wird eine Label Map benötigt, die es ermöglicht, gefundene Objekte zu benennen.

Ordnerstruktur in der Cloud: Um das Training in der Cloud durchführen zu können müssen die Daten in die Cloud geladen werden. Die Struktur der benötigten Dateien sieht dann wie folgt aus :

```
YOUR_GCS_BUCKET
├── data
│   ├── Konfigurationsdatei
│   ├── model.ckpt.index
│   ├── model.ckpt.meta
│   └── model.ckpt.data-00000-of-00001
```

```

├── face_label_map.pbtxt
├── train.record-*.
└── val.tfrecored-*.

```

5.2 Training in der Cloud

Für das Training in der Google Cloud mussten die Bilddaten in einen sequenziell auslesbaren Datenstrom umgewandelt und in der Cloud gespeichert werden. Der Aufbau der Infrastruktur, beziehungsweise welche Grafikkarten für das Training verwendet werden sollen, wird in der Cloud.yml definiert. Zudem muss das Objekterkennungsmodell in einer Konfigurationsdatei definiert werden. In dieser Arbeit werden die Modelle **faster_rcnn_nas** und **ssd_mobilenet_v2_coco** getestet. Die beiden Modelle wurden auf dem COCO-Datensatz vortrainiert ([11](#)).

Das **faster_rcnn_nas** ist ein Modell, welches für die Objekterkennung verwendet wird. Es ist jedoch nicht so schnell, wie das **ssd_mobilenet_v2_coco** ([11](#)).

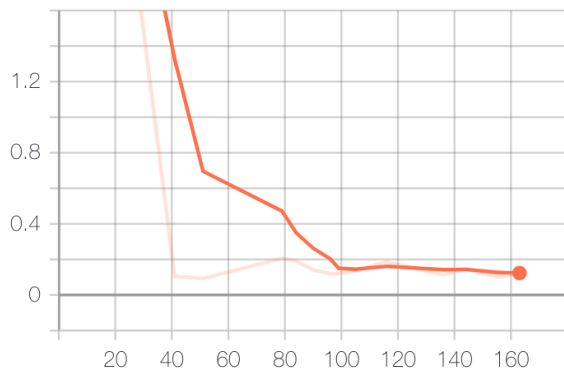
5.3 Ergebnisse

Zur Prüfung des Lernerfolgs beim Training wird der Parameter Total Loss verwendet. Der Total Loss setzt sich aus der Summe des Classification Loss und des Localization Loss zusammen. Der Classification Loss gibt die Genauigkeit der Objektklassifizierung an.

Wurde bei der Klassifizierung ein Gesicht gefunden, wird der Localization Loss berechnet. Dieser gibt an, wie genau die lokalisierte Position zu der korrekten Position des Trainingsbildes ist (Vergleich der Koordinaten der Diagonalen).

In den Abbildungen [5.1](#) bis [5.4](#) sind die Prüfparameter Total Loss, Classification Loss und Localization Loss des Trainings für die zwei Modelle dargestellt, wobei die Ausgangswerte (dünne Linien) geglättet wurden (dicke Linien). Auf der X-Achse ist die Zahl der ins Training einbezogenen Bilder dargestellt.

BoxClassifierLoss/classification_loss
tag: Loss/BoxClassifierLoss/classification_loss



BoxClassifierLoss/localization_loss
tag: Loss/BoxClassifierLoss/localization_loss

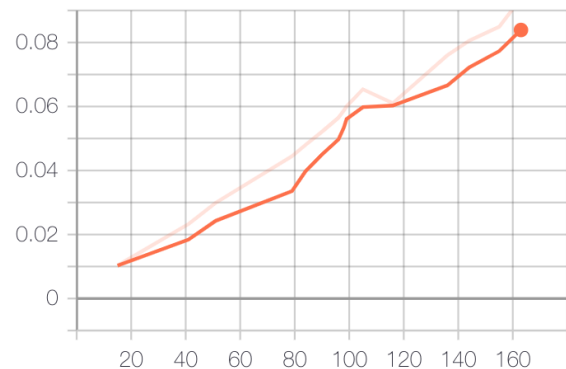


Abbildung 5.1: Die beiden Abbildungen zeigen den Classification Loss und den Localization Loss des *rcnn_nas_coco*-Models

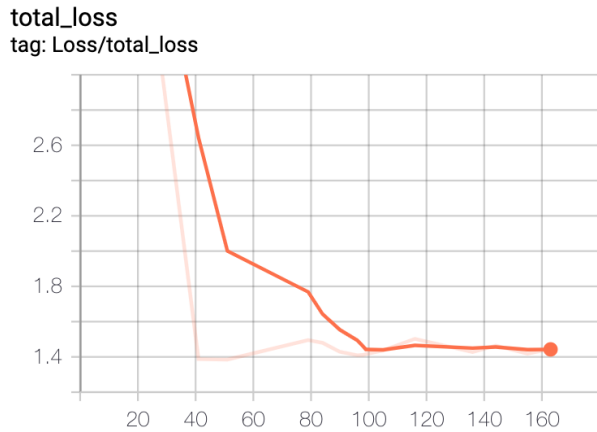


Abbildung 5.2: Der Total Loss für das „rcnn_nas_coco“-Model

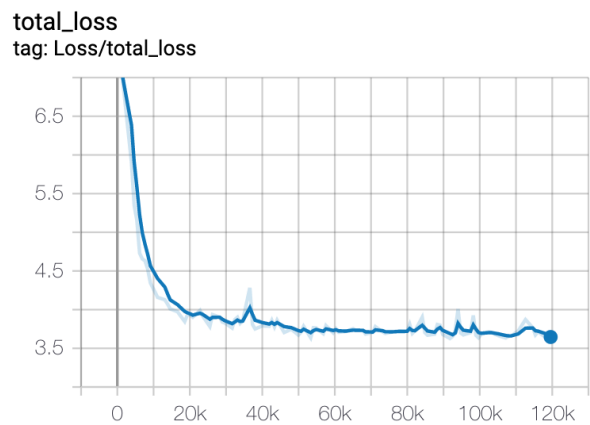


Abbildung 5.3: Der Total Loss für das „ssd_mobilenet_coco“-Model

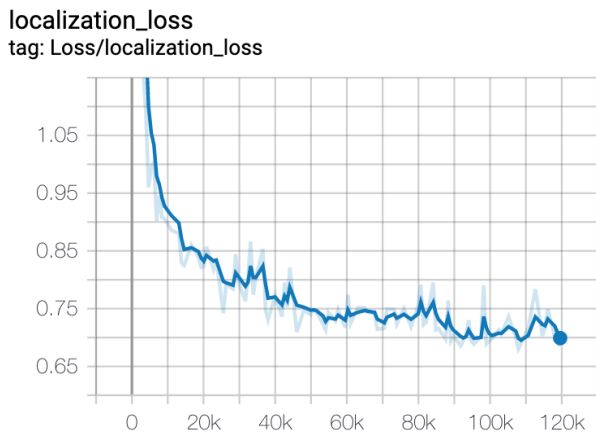
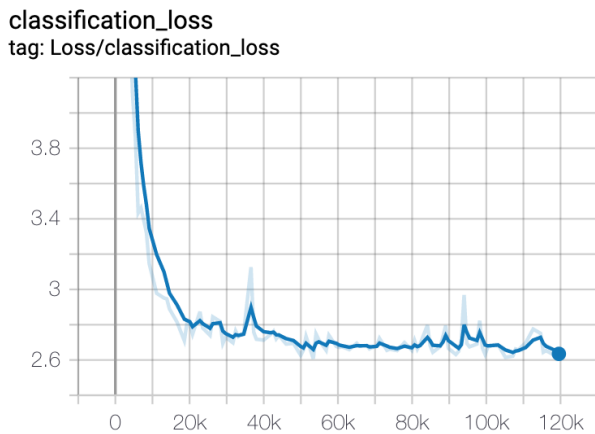


Abbildung 5.4: Die beiden Abbildungen zeigen den Classification Loss und den Localization Loss des „ssd_mobilenet_coco“-Models

Die Ergebnisse der beiden Modelle ist nach Bewertung der Verlustfunktionen eindeutig (Total Loss): Das *rcnn_nas_coco*-Model erreicht wesentlich niedrigere Total Loss Werte als das *ssd_mobilenet_coco*-Model, ist also wesentlich genauer. Der Unterschied kommt zustande, obwohl die Trainingsdauer bei dem *rcnn_nas_coco*-Model wesentlich länger (23 Stunden) ist und nur 160 Bilder einbezogen wurden, während das *ssd_mobilenet_coco*-Model nach 23 Stunden mit 120.000 analysierten Bildern beendet wurde.

5.4 Testphase

Bei der Testphase habe ich mich für das *ssd_mobilenet_coco* Model entschieden, obwohl die Total Loss Werte aus dem Training geringer waren. Diese Entscheidung wurde getroffen, da das andere Model den ausführenden Laptop zum Absturz gebracht hat. Die Tests wurden anhand von 5 Bildern aus dem WIDER_test Datensatz, mit der in Kapitel 6 beschriebene Applikation, durchgeführt und dabei geprüft ob das trainierte Model Gesichter erkennen kann. Die Bilder sind im folgenden aufgeführt und enthalten 34 Gesichter wovon 32 Gesichter gefunden wurden, wobei davon doppelt gefundene Gesichter nicht beachtet werden. Zu dem hat das Model zwei weitere Gesichter gefunden wo keine zu sehen waren.

Bild	Beschreibung
 <p>Download from Dreamstime.com 30954086 Dmitry Shironosov Dreamstime.com</p>	Es wurden 4 von 4 Gesichter gefunden
	Es wurden 4 von 4 Gesichter gefunden

Tabelle 5.1: Bilder die mit der Object detection API auf Gesicht untersucht wurden

Bild	Beschreibung
	<p>Es wurden 16 von 18 Gesichter gefunden.</p>
	<p>Es wurden 4 von 4 Gesichter gefunden.</p>
	<p>Es wurden 4 von 4 Gesichter gefunden.</p>

Tabelle 5.1: Bilder die mit der Object detection API auf Gesicht untersucht wurden

Zusammenfassend wird festgestellt, dass trotz der hohen Total Loss gegen über dem `ssd_mobilenet_coco`-Mode 94% der Gesichter gefunden wurden.

6

Die Applikation

Die Applikation wurde mit Tensorflow und OpenCV Entwickelt und ist unter dem Link <https://github.com/DevJakobL/facedetectionapp> zu finden.

6.1 Projektaufbau

```
facedetectionapp
└─ data
```

In dem **facedetectionapp** befindet sich die Applikation zum Erkennen von Gesichtern. Die für das Erkennen von Gesichtern zuständigen neuronalen Netzen können heruntergeladen werden und müssen im dem Ordner **data** gespeichert werden.

6.2 Arbeiten mit dem Applikation

Um die Facedetection Applikation verwenden zu können und wie diese aufgerufen werden kann, wird im folgenden erläutert:

```
app.py [-h] --image IMAGE --frozen FROZEN [--output OUTPUT] [--tb TB] [--
        overview OVERVIEW]
```

Mit dem Pflichtparameter **–image** wird das zu untersuchende angegeben. Zur Angabe des trainierten Netzwerkes wird der Parameter **–frozen** verwendet. Dieser erwartet Datei mit einem Frozen Graph. Dieser kann nicht weiter trainiert werden und wird dazu verwendet eine Trainiertes Neuronale Netz in einer Applikation zu verwenden. Mit dem Optionalen Parameter **–output** wird der Speicherordner angegeben wo die bei der Analyse gefundenen Gesichter gespeichert werden. Mit dem optionalen Parameter kann angegeben werden ob das Netz mit TensorBox Trainiert wurde. Des weiteren kann mit dem Parameter **–overview** speichern kann zusätzlich ein Image mit allen gefundenen Gesichtern gespeichert werden.

6.3 Wichtige Funktionen

Funktion	Beschreibung
<code>getImageName(image)</code>	Hilfsfunktion die den Namen des Bildes aus dem übergebenen Pfad berechnet

Tabelle 6.1: Funktionen innerhalb der Applikation

Funktion	Beschreibung
<code>save_images(image, output, name, count=None)</code>	Hilfsfunktion zum Speichern von Bildern. Dazu wird das Bild, der Pfad zum Bild und der Bildname angegeben. Zudem kann mit dem optionalen Parameter Count angegeben werden, wievielte Gesicht zum Zeitpunkt des Speicherns gefunden wurde.
<code>load_frozen_graph(frozen_graph_filename)</code>	Dies Hilfsfunktion wird verwendet den Frozen Graph zu laden. Die übergebene Datei enthält das Neuronale Netz und die Gewichtungen der einzelnen der Knotenpunkte der Neuronen.
<code>object_detection(args)</code>	Diese Funktion ist die Hauptfunktion und übernimmt das Analysieren der Bilder und bekommt die Argumente aus der Konsole übergeben.

Tabelle 6.1: Funktionen innerhalb der Applikation

7

Zusammenfassung und Ausblick

Im Verlauf der Arbeit wurden das Tool TensorBox und die Objekt detection API hinsichtlich ihrer Fähigkeit, für die Gesichtserkennung trainiert zu werden, untersucht. Bei der Verwendung des Tools TensorBox wurde festgestellt, dass dieses nicht für den weiteren

Gebrauch empfohlen werden kann, da dieses Tool nicht weiterentwickelt wird. Ein weiterer Grund dafür, dass sich Tensorbox nicht eignet, ist, dass es nicht gut dokumentiert ist und es dadurch schwierig ist, es in der eigenen Entwicklungsumgebung zum Laufen zu bringen. Auch die Ergebnisse des Tests waren nicht sehr überzeugend, da nur wenige Gesichter gefunden werden konnten und nicht nur unscharf abgebildete Gesichter, sondern auch Gesichter in Frontalansicht bei guter Ausleuchtung nicht erkannt wurden.

Die Verwendung von Object detection API ist zu empfehlen, da diese weiterentwickelt wird und eine gute Dokumentation vorliegt. Des Weiteren hat sie den Vorteil, dass sie ein Teil von TensorFlow ist und sie von Google selbst verwendet wird. Die Object detection API lässt sich auch einfach auf der Google Cloud AI-Plattform verwenden, da sie dort integriert ist. Ein Nachteil der Nutzung der Cloud ist, dass es mit erheblichem finanziellen Aufwand verbunden ist. Deshalb ermöglicht es die API auch sein Neuronales Netz lokal zu Trainieren. Die Ergebnisse des Neuronalen Netzwerkes unter der Object Detection API zeigen auch wesentlich bessere Ergebnisse, da fast alle Gesichter auf den Testbildern gefunden wurden, allerdings manchmal überlappend.

Die Entwickelte Applikation Ermöglicht es dem Nutzern Gesichter auf Bildern zu finden und auszuschneiden um sie weiter zu verarbeiten. Für die Suche nach Gesichtern kann die Applikation Neuronale-Netze verarbeiten die mit TensorBox und der Object detection API trainiert wurden.

In weiteren Projekten ist es möglich, die Applikation so weiter zu entwickeln, dass sie Gesichter, die sie doppelt findet, nur einmal auszuschneiden. Des Weiteren bildet diese Arbeit eine Grundlage zur Implementierung von einer Software, die Gesichter erkennt und sie einer bestimmten Person zuweist (Face Recognition). Auch das Testen und Trainieren von weiteren Objekterkennungsmodellen von der Objekt Detektion API bildet eine Möglichkeit für weitere Verbesserungen der Ergebnisse und eine Grundlage für weitere Projekte.

Die Applikation kann auch so Erweitert werden, dass Gesichter in Videos erkannt werden können.

Auch das Testen und Trainieren von weiteren Objekterkennungsmodellen von der Objekt Detektion API bildet eine Möglichkeit für weitere Verbesserung der Ergebnisse und eine Grundlage für weitere Projekte.

Literatur

- (1) Russell Stewart. *TensorBox: A Fast Object Detection Framework in TensorFlow*. C, 2016.
- (2) Russell91. *Russell91/TensorBox: Object detection in TensorFlow. Brought to you by Kuperl Industries*. 2018. URL: <https://github.com/Russell91/TensorBox> (besucht am 25.08.2019).
- (3) Jonathan Huang u. a. “Speed/accuracy trade-offs for modern convolutional object detectors”. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. Bd. 2017-Janua. Nov. 2017, S. 3296–3305. ISBN: 9781538604571. DOI: [10.1109/CVPR.2017.351](https://doi.org/10.1109/CVPR.2017.351). arXiv: [1611.10012](https://arxiv.org/abs/1611.10012). URL: <http://arxiv.org/abs/1611.10012>.
- (4) Google Inc. *models/research/object_detection at master · tensorflow/models*. URL: https://github.com/tensorflow/models/tree/master/research/object_detection (besucht am 25.08.2019).
- (5) Shuo Yang u. a. “WIDER FACE: A face detection benchmark”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Bd. 2016-Decem. 2016, S. 5525–5533. ISBN: 9781467388504. DOI: [10.1109/CVPR.2016.596](https://doi.org/10.1109/CVPR.2016.596). URL: <http://shuoyang1213.me/WIDERFACE/>.
- (6) Russell Stewart. *Annotation data formats*. URL: <https://github.com/Russell91/TensorBox/blob/master/utils/annolist/readme.md> (besucht am 24.08.2019).
- (7) Ming-Hsuan Tu. *twmht/python-widerface: Simple WIDERFACE data parser written in python*. URL: <https://github.com/twmht/python-widerface> (besucht am 24.08.2019).
- (8) Nils Brinkmann, René Grzeszick und Dr-Ing A Gernot Fink. “Objekterkennung in natürlichen Szenen mittels Region-based Convolutional Neural Networks”. In: (2016). URL: <http://www.cs.uni-dortmund.de>.
- (9) Ivan Itzcovich. *iitzco/widerface-to-tfrecord: Convert WIDER Face Dataset to Tensorflow's TFRecord format*. 2018. URL: <https://github.com/iitzco/widerface-to-tfrecord> (besucht am 24.08.2019).
- (10) Google Inc. *models/using_your_own_dataset.md at master · tensorflow/models*. URL: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/using_your_own_dataset.md (besucht am 24.08.2019).
- (11) Google Inc. *models/detection_model_zoo.md at master · tensorflow/models*. 2019. URL: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md (besucht am 25.08.2019).