Reference Manual ToolChain

Generated by Doxygen 1.8.13

Contents

Chapter 1

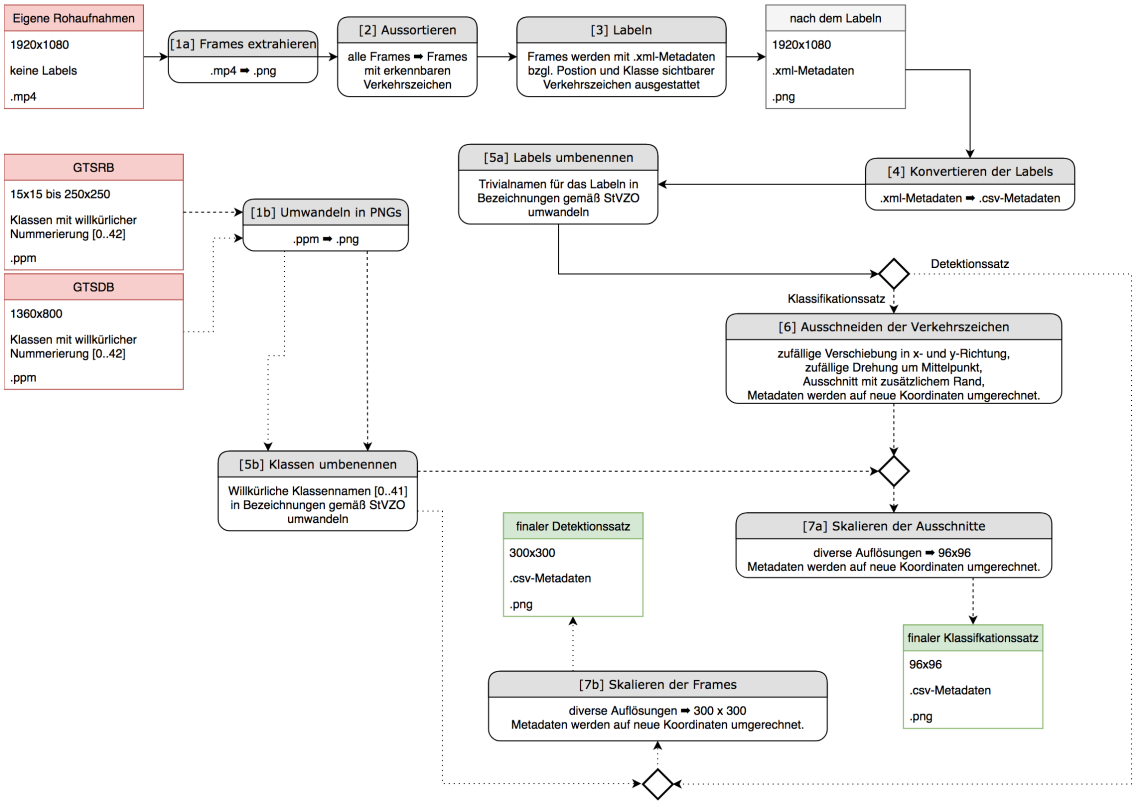
Einleitung

Im Folgenden wird der Prozess für das Erstellen eines Datensatzes, welcher für die Verwendung mit der entwickelten API / App geeignet ist, erklärt.

Außerdem finden Sie hier eine Anleitung, wie die Neuronalen Netzwerke zur Detektion und Klassifikation trainiert und benutzt werden könnten.

Chapter 2

Tool-Chain



[1a] Es werden einzelne Frames aus den Videodateien extrahiert, um diese Frame für Frame zu labeln.
Skript aus ToolChain/Labeling/

```
video_to_frames.py -i input_video.mp4 -o output_folder/
```

[1b] Da sowohl der verwendete Datensatz GTSDB als auch GTSRB Bilder im .ppm-Format vorliegen, werden diese in .png umgewandelt.
Dies geht mit dem Skript aus ToolChain/Dataset:

```
ppm_to_png.py -i input_folder -o output_folder
```

[2] Manuell werden Frames aussortiert, auf welchen keine gut sichtbaren Verkehrszeichen vorhanden ersichtlich sind. Diese können in Bereichen angegeben werden. Wenn sich z.B. auf den Bildern 203 bis 250 keine VZ befinden kann dies in eine CSV Datei eingetragen werden, welche sich in dem in Schritt **[1]** erstellten Unterordner befindet.

Wenn man alle Bereiche angegeben hat, kann man diese mit dem Skript aus `ToolChain/Labeling/` wie folgt löschen:
`delete_from_csv.py -i output_folder`
output_folder ist der Ordner aus Schritt **[1]**

[3] Mit der Software *"LabelImg"* wird der `output_folder` (Schritt **[1]**) geöffnet. Dort muss für jedes Bild eine rechteckige Bounding-Box angelegt werden und die korrekte Klasse ausgewählt werden. Diese Klassen befinden sich in: `ToolChain/Labeling/predefined_classes.txt`

[4] Die Metadaten, welche zuvor mit LabelImg erstellt wurden, werden für eine einfacherer Handhabung von einzelnen .xml-Dateien (Beim Bearbeiten von LabelImg erstellt worden) in eine CSV-Datei umgewandelt. Dies ist mit dem Skript aus `ToolChain/Dataset` wie folgt realisierbar:
`xml_to_csv.py -i input_folder_with_xml -o output.csv -o output_folder`

[5a] Da in Schritt **[3]** die Klassen mit Trivialnamen benannt wurden und dies teilweise unübersichtlich ist, werden die Klassenamen in ihre Beziechung nach StVZO umbenannt. Dies geht mit dem Skript aus `ToolChain/Dataset`:
`translate_by_csv.py -i labels_old.csv -d dict.csv -c class -o labels_new.csv`
Hierbei ist dict.csv im Ordner `ToolChain/Dataset/CSVs/` eine CSV, welche die entsprechende Umbenennung kennt.

[5b] Die gleiche Problematik wie in **[5a]** tritt auch bei den GTS-Datensätzen auf. Die Übersetzung kann allerdings direkt mithilfe des ppm_to_png-Skriptes durchgeführt werden. Dafür müssen nur zusätzlich der Parameter `-dict` mit dem Pfad zu einer Dictionary-CSV befriedigt werden, so wie das Rekursiv-Flag (`-r`) gesetzt werden um auch Unterordner einzubeziehen.

[6] Für den Klassifikationsdatensatz, müssen die Verkehrszeichen aus den großen Bilderen ausgeschnitten werden und auf eine Größe von 96x96 skaliert werden. Hierbei werden die gleichen Kriterien wie beim verwendeten Datensatz GTSRB eingehalten. Dieses besteht daraus, dass die Bounding-Box entweder 5 Pixel oder 10% der Breite, je nachdem was größer in alle Richtungen erweitert. Um ledigliche die Verkehrszeichen aus den Bildern auszuschneiden, muss folgendes Skript aus `ToolChain/Dataset` ausgeführt werden:
`extract_sign_from_image.py -i input_folder -l labels.csv`

Um den Datensatz zu erweitern, wurde mit einem weiteren Skript eine leichte Rotation und eine leichte Verschiebung der Bounding-Box eingeführt. In dem im folgenden verwendeten Skript lassen sich diese Parameter um Zeile 439 Parameter einstellen. Dies gelten als obere Grenze und es wird jedes Bild, mit zufälligen Werten, welche von besagten Parametern begrenzt werden, acht weitere Bild generiert. Dies erlaubt das Skript aus `ToolChain/Dataset`: ´rotate_and_offset.py -i input_folder -l labels.csv -o output_folder‘

[7a][7b] Für die Detektion müssen die Bilder auf 300x300 herunterskaliert werden, da das verwendete Neuronale Netzwerk (*SSDLite*) dies als Eingabegröße besitzt. Dies kann mit dem Skript aus ToolChain/Dataset getan werden: `resize_dataset.py -i input_folder -l labels.csv -o output_folder -s scale_size` Wobei scale_size in diesem Fall 300 wäre, damit das Bild auf 300x300 skaliert wird.
Hierbei wird das Bild verzerrt, da kein *Zero-Padding* angewandt wird.

Chapter 3

Training des Detektionsnetzwerks

Zum Training wird eine angepasste Version der TensorFlow Object Detection API benutzt. Um das Training zu beschleunigen werden vortrainierte Gewichte benutzt.
(siehe https://github.com/tensorflow/models/blob/master/research/object_↩ detection/g3doc/detection_model_zoo.md)

Diese befinden sich im Ordner `NeuralNetwork/Detection/object_detection/SSDLite/`.

Trainingsparameter wie z.B. die Lernrate können in der vorhandenen `pipeline.config` geändert werden.

Außerdem muss der Datensatz erneut umgewandelt werden, da TensorFlow diesen in einem speziellen Format benötigt. Das Skript unter `NeuralNetwork/Detection/object_detection/SSDLite/utils` ermöglicht dies mit folgender Kommandozeileneingabe:
`python generate_tfrecord.py --csv_input=labels.csv --output_path=output.↩ record`

Hierbei müssen in der `labels.csv` die entsprechenden Metadaten enthalten sein. Außerdem müssen die Übersetzungen in Klassennummern ab Zeile 30 ggf. angepasst werden. In Zeile 90 muss noch der Ordner mit den Bildern, welche in einer Größe von 300x300 vorliegen angegeben werden.

Die generierten `test.record` bzw. `training.record` Dateien müssen in den Ordner `Neural↩ Network/Detection/object_detection/data/` kopiert werden. Eventuell sind hier Anpassungen in der `pipeline.config` nötig.

Anschließend kann das Trainig aus dem NeuralNetwork/Detection/object_detection/ Ordner wie folgt gestartet werden:
´python train.py –logtostderr –train_dir=training_dir/ –pipeline_config_path=SSDLite/pipeline.config´

Hierbei werden die Trainigsparameter aus der pipeline.config benutzt und die verschiedenen checkpoints in dem training_dir/ gespeichert.

Der Verlauf des Trainings kann wie folgt angezeigt werden:
`tensorboard --logdir='training_dir/`

Dies öffnet einen Webserver auf *Port 6006*, auf welchem die Graphen dargestellt werden können.

Wenn man sich für einen Checkpoint entschieden hat, kann dieser in eine *Protobuf-Datei* umgewandelt werden, welche für die App verwendet werden kann. Dies kann über das Skript aus dem Ordner `NeuralNetwork/↩ Detection/` wie folgt ausgeführt werden:
`python object_detection/export_inference_graph.py --input_type image_tensor`

```
--pipeline_config_path object_detection/SSDLite/pipeline.config --trained↩
_checkpoint_prefix object_detection/training/model.ckpt-nummer --output_↩
directory object_detection/output/
```

Hierbei wird erneut die `pipeline.config` verwendet, da die auch Allgemeine Infos über das Netzwerk beinhält. Außerdem wird bei `--trained_checkpoint_prefix object_detection/training/model.↩ ckpt-nummer` die Nummer durch den entsprechenden checkpoint ersetzt z.B. model.ckpt-43456 Nach ausführung des Skriptes liegt im Ordner `NeuralNetwork/Detection/object_detection/output/` eine Datei names `frozen_graph.py`, welcher das Netzwerk in dem Format beinhaltet, so dass es von der App verwendet werden kann.

Chapter 4

Training des Klassifikationsnetzwerks

Zum Trainieren des Klassifikator muss die Datei `NeuralNetwork/ Classification/mobilenetv2.py` per Kommandozeile aufgerufen werden.

Das Trainingsskript musste sehr kurzfristig umgestaltet und auf das Framework *Keras* umgebaut werden, da ein nicht näher klärbarer Bug in TensorFlow aufgetreten ist. Daher war es bisher nicht möglich das Skript komfortabel mit Kommandozeilenparametern auszustatten. Stattdessen befinden sich diese im Kopf des Skripts. Die Wichtigsten sollen im Folgenden kurz erklärt werden:

Zum einen die Pfade zu den Datensätzen zum Training bzw. zur Validierung:

```
train_data_path = '/datasets/swp2018/split_img/train'
validation_data_path = '/datasets/swp2018/split_img/validation'
```

Außerdem die Anzahl der Trainingsdateien sowie deren Größe und die Anzahl der Klassen:

```
file_count_train = 134443
image_size = 96
classes = 47
```

Desweiteren die für Neuronale Netzwerke üblichen Parameter:

```
epochs = 200
batch_size = 128
learn_rate = 0.045
decay = 0.00004
momentum = 0.9
```

Sowie den für MobileNet spezifischen Alpha-Parameter welcher die Anzahl der Filter pro Schicht ändert. Die genauere Bedeutung/Verwendung kann aus dem entsprechenden Paper unter `https://arxiv.org/pdf/1704.04861.pdf` entnommen werden:

```
alpha_val = 0.35
```

Nach dem Abschluss einer Epoche werden die Gewichte des Netzwerks automatisch gespeichert. Beim Beenden des Trainings soll aus diesen das Optimum (anhand selbst festgelegter Kriterien) ausgewählt werden.
Zur Umwandlung dieser Datei in das TensorFlow-Datenformat kann der Befehl

```
python keras_to_tensorflow.py -input_model weights.hdf5
```

genutzt werden. Wobei `weights.hdf5` die gewünschte Gewichts-Datei darstellt.