OpenImages Dataset in TensorFlow Object Detection trainieren

1. Dataset von OpenImages herunter laden

Mit dem OIDv4_ToolKit können alle in OpenImages vorhandenen Klasses heruntergeladen werden. Dafür die gewünschten Klassen entweder Komma getrennt oder in einem Text File demm --classes Argument übergeben.

```
python3 main.py downloader --Dataset Dataset_Folder/ --classes classes.txt
--limit 2000 --type_csv all
```

'Dataset_Folder' muss in Ordner OID zusammen mit einem Ordner csv_folder liegen.

in csv_folder muss die Dateien train-annotations-bbox.csv, test-annotations-bbox.csv, validation-annotations-bbox.csv und class-descriptions-boxable.csv enthalten.

Da Klassen Namen in den Label Files keine Leezeichen habe dürfen sollten diese durch '_' ersetzt werden. Dafür kann folgendes Script verwendet werden:

```
python3 remove_spaces.py path/to/folder
```

2. Bilder aus Unterordnern in test/train/validation Ordner schreiben

Nun müssen die Bilder aus den jeweiligen unterordnern in train bzw. test Ordner verschoben werden. Dafür das Script 1_collect_images_into_one_folder.sh in DATASET_DIR kopieren und ausführen:

```
./collect_files_from_subfolders.sh
```

3. Labels Files konvertieren

3.1 OpenImages zu Pascal VOC

Zunächst müssen die Label files vom OpenImages ins PASCAL VOC Annotations Format konvertiert werden.

```
python3 oi_to_pascal_voc_xml.py \
    --dataset_path DATASET_DIR/
```

DATASET_DIR muss dabei train und test Ordner enhalten.
Diese müssen alle Bilder und einen Ordner 'labels' mit allen label.txt files enthalten.

3.2 Pascal VOC zu CSV

Anschließend die erhaltenen XML Files in jeweils für train und test eine CSV Datei schreiben.

```
python3 pascal_voc_xml_to_csv.py -i DATASET_DIR/
```

Wird automatisch für train und test Ordner in DATASET_DIR Ordner ausgeführt.

3.3 CSV zu TF Records

Beide CSV Files *train.csv* und *test.csv* können jetzt in TF Record Files convertiert werden.
Diese enthalten auch die Bilder sind dann das komplette Datenset für das Training.

```
python3 csv_to_tf_record.py  \
    --csv_input=DATASET_DIR/train.csv \
    --image_dir=DATASET_DIR/train/ \
    --classes=DATASET_DIR/classes.txt
```

muss für train und test separat ausgeführt werden.

4. Training vorbereiten

4.1 Label Map

Datei **label_map.pbtxt** erstellen und mit den Klassen und deren Index befüllen.
Dafür folgendes Script mit classes.txt in dem alle vorhandenen Klasses gelistet sind ausführen.

```
python3 create_label_map.py classes.txt
```

Ordner **data** anlegen und **train.record**, **test.record** und **label_map.pbtxt** verschieben.

4.2 vortrainierten Model herunterladen

Tensorflow Object Detection Model Zoo

4.3 Config File

Das für das Model richtige Config File herunteer laden und folgende Stellen anpasse:

```
num_classes # anzahl der Klassen
```

```
fine_tune_checkpoint "path/to/model_folder/model.ckpt" # pfad zum
heruntergeladenen Model
```

```
train_input_reader: {
  tf_record_input_reader {
    input_path: "data/train.record"
  }
  label_map_path: "data/label_map.pbtxt"
}
# sowie für eval_input_reader
```

```
eval_config: {
  metrics_set: "coco_detection_metrics"
  num_examples: N_TEST  # entspricht Zeilenzahl aus test.csv
}
```

und anschließend auch in **data** Ordner verschieben.

5 Training

5.1 Training starten

```
python3 paht/to/models/research/object_detection/model_main.py \
--pipeline_config_path=data/MODEL_CONFIG_FILE.config \
--model_dir=OUTPUT_DIR \
--alsologtostderr \
--num_train_steps=N_STEPS_TO_TRAIN
```

Falls Memory Error Auftreten die Batch Size im Config File veringern und erneut versuchen.

5.2 Visualisierung

```
tensorboard --logdir OUTPUT_DIR
```

5.3 Trainierten Tensorflow Graph Exportieren

```
python3 path/to/models/research/object_detection/export_inference_graph.py
\
    --input_type image_tensor \
    --pipeline_config_path data/MODEL_CONFIG_FILE.config \
    --trained_checkpoint_prefix OUTPUT_DIR/model.ckpt-NR \
    --output_directory DATASET_DIR/frozen_graph/
```

erzeugt einen Ordner *frozen_graph* der *frozen_inference_graph.pb* enthält.
Kann jetzt mit **OpenVino Model Optimizer** für **InferenceEngine** konvertiert werden

10. TensorFlow Graph für OpenVino konvertieren

Der Exportierte TensorFlow Graph kann nun mit dem Model Optimizer von OpenVino für die Inference Engine konvertiert werden.

Dafür folgenden Command ausführen:

```
python3 /opt/intel/openvino/deployment_tools/model_optimizer/mo_tf.py \
--input_model path/to/frozen_inference_graph.pb \
--output_dir path/to/export/model \
--tensorflow_use_custom_operations_config model_support.json \
--tensorflow_object_detection_api_pipeline_config path/to/pipeline.config \
--input_shape [1,h,w,3] \
--data_type FP16 \
--input image_tensor \
--output=detection_classes,detection_scores,detection_boxes,num_detections
\
--reverse_input_channels
```

Je nach model das entsprechende model_support.json file aus */opt/intel/openvino/deployment_tools/model_optimizer/extensions/front/tf/* verwenden.
Für SSD Modelle die in Tensorflow v1.14 trainiert wurden **ssd_support_api_v1.14.json** (wenn nur konvertiert werden soll, also ohne training: ssd_support.json nehmen)

input_shape aus config file übernehmen. data_type für NCS2 immer FP16

Die dadurch erzeugten xml und bin files können nun für die Inferenz mit Open Vino verwendet werden.