Using Google's API Object Detection for Transfer Learning

January 29, 2020

Step 0 - API installation

Clone Google's Tensorflow API repo : https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md.
Then in the folder 'models/research/':

```
python3 setup.py install
```

IMPORTANT: every time one wants to use the API, the folders 'tensorflow/models/research/' et 'slim' must be added to the environment variable PYTHONPATH. This can be done with the following line, executed in the folder tensorflow/models/research/:

```
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

To avoid doing that at each use, one can add this instruction in the .bashrc file so that the instruction will be executed whenever a new terminal is opened.

Step 1 - Getting a pre-trained model

Google provides pre-trained models on classic dataset like COCO, Open Images or Kitti. State-of the art architectures can be found (Faster R-CNN, SSD MobileNetv2, Mask R-CNN, ...).
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

The files found in a downoladed model are the following :

1. pipeline.config : files which contains the parameters chosen for the first training of the model (Loss function, Optimizer, data augmentation, ...). This file must be used again for the Transfer Learning, the paths must be changed to the one's of the host computer.

2. model.ckpt.xxxx : composed of three files. They are used to recover the weights of the pre=trained model. These files will permit to perform the Transfer Learning by starting the training from these weights for the new model.

3. frozen_inference_graph.pb : file that is used to do inference with the model.

Step 2 - Preparing the inputs of the training

This step is just a suggestion for the organization one could use for the training.
Create two folders :

1. **model** - this folder will contain the pipeline.config file.
   As mentioned earlier, copy the pipeline.config file from the pre-trained model and modify

it. At least, modify the number of classes of the new model (how many objects classes your dataset has) and the path where are located the checkpoint file .ckpt ('fine_tune_chekpoint'), the label map (contains the list of objects classes) abd the files train.record and eval.record (see Step 4). The paths to modify are indicated by 'PATH_TO_BE_CONFIGURED' in the file.
Finally, it is recommanded to add the line 'from detection checkpoint: true' in the 'train_config' part, after the line 'fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED"'.

2. **data** - this folder will contain the custom dataset you have. The dataset needs to be split into two folders : eval and train. Typically, the training set contains 80 to 90% of the images, the rest goes into the eval set.

Step 3 - Building or using a dataset

Get a dataset containing labeled images of the object you want the network to detect, or build your own dataset using an annotation. Many annotation software are available, for example VOTT from Microsoft.
Be sure to export the annotation file in the .csv format, one csv file for the training set and one for the eval set.

Step 4 - Generating TFRecord files

Both .csv files generated before need to be converted into a specific format for TensorFlow, .record. This format file is just a format developed by Google for TensorFlow to store information.
The script 'generate_tfrecord.py' available on GitHub (https://github.com/simheo/transfer_learning.git) permits to convert an annotation file .csv into the a tfrecord file. In the script, both the names and number of classes need to be changed according to the dataset used.
Three flags must be specified to execute the program : –csv_input (path to csv file), –image_dir (path to the set of images, training or eval set) and –output_dir (name of the file we want, train.record ou eval.record, to be generated and path to where it should be put).

Paths for both eval.record and train.record should be added to pipeline.config file.

NOTE: in the script 'generate_tfrecord.py', a number is associated to each class name. Be careful not to modifiy these numbers between the generation of the two .record files. The number associated to each class must be identical in eval.record and train.record.

Step 5 - Training the model

(before the training, don't forget execute the line for PYTHONPATH specified in Step 0)

In the folder 'models/research' of the cloned repo:

```
python object_detection/model_main.py \
    --pipeline_config_path=PATH_TO_PIPELINE_FILE_IN_/MODEL \
    --model_dir=PATH_TO_MODEL/model_trained \
    --num_train_steps=NB_STEPS \
    --sample_1_of_n_eval_examples=1 \
    --alsologtostderr
```

model_dir corresponds to the path where you want the files generated by the training to be stored.

NB: the training can be monitored using tensorboard:

```
tensorboard --logdir=${MODEL_DIR}
```

with model_dir the same pathspecified above.
One this instruction executed, a port is opened to access the tensorboard's interface (typically with a web browser go to localhost:6006). Tensorboard allows to monitor the evolution of the loss value and the mAP (Mean Average Precision, metrics used to measure the precision of the model)

Step 6 - Exporting the model

The model once trained cannot be used directly for inference. To permit inference, another script of the API must be used.
Once again, in the repo 'model/research'

```
python object_detection/export_inference_graph.py \
    --input_type=image_tensor \
    --pipeline_config_path=PATH_TO_PIPELINE_FILE_IN_/model \
    --trained_checkpoint_prefix=PATH_TO_MODEL/model_trained/model.ckpt-XXXXX \
    --output_directory=PATH_TO_MODEL/model_export
```

output_directory correponds to the path that will be created after the instruction and containing the exported model.

NOTE: it seems that a graph can only be exported if the version of Tensorflow used to export is the same as the version for training.

Step 7 - Using the model for inference

The model that an be used for inference can be found in the file **'frozen_inference_graph.pb'**.
In the Github's repo cloned at Step 0, Google provides a Jupyter Notebook that can be used to do inference on images with the new trained model. The notebook can be found in 'research/object_detection', it might give good leads on how to use the API.
A notebook to do inference on a video stream is available on GitHub: https://github.com/simheo/transfer_learning.git

BONUS - resume a training from a checkpoint

It is possible to continue a training that has been stopped or has finished but has not converged yet (cf monitoring using Tensorboard).
To continue the training where it stopped, use step 5 again, but with two modifications in the pipeline.config file :

- modifier la ligne **'fine_tune_checkpoint'** where to usually specify the path to the pre-trained model. Replace this path by the path where the the checkpoint of the new trained model is.
- after the line 'fine_tune_checkpoint', add the instruction

```
    load_all_detection_checkpoint_vars: true
```

The remaining of the training procedure remains the same.