Single Shot MultiBox Detector in TensorFlow

tensorflow ssd deep-learning yolo object-detection

96 commits 1 branch 0 releases 2 contributors

Branch: master New pull request Find file Clone or download

balancap FIX: Fine tuning of ImageNet models, adding checkpoint scope parameter. Latest commit 6eb6e74 9 days ago

checkpoints	ADD: SSD 300 TF checkpoints and demo images.	2 months ago
datasets	UPDATE: Several improvements to Evaluation script.	26 days ago
demo	ADD: SSD 300 TF checkpoints and demo images.	2 months ago
deployment	CLEAN: Training script and model_deploy.py	2 months ago
nets	FIX: Small bugs in Training and Eval scripts.	18 days ago
notebooks	UPDATE: Small modification to SSD select method.	19 days ago
pictures	UPDATE: Readme and SSD Notebook.	24 days ago
preprocessing	UPDATE: SSD-512 model. Basically working.	22 days ago
tf_extended	UPDATE: Readme - Training part.	18 days ago
.gitignore	UPDATE: ignore file.	10 days ago
COMMANDS.md	FIX: Fine tuning of ImageNet models, adding checkpoint scope parameter.	9 days ago
README.md	FIX: Fine tuning of ImageNet models, adding checkpoint scope parameter.	9 days ago
caffe_to_tensorflow.py	FIX: Caffe to TensorFlow script, number of classes.	20 days ago
eval_ssd_network.py	[FIX]eval_ssd_network some flags to right type	17 days ago
inspect_checkpoint.py	FIX: Fine tuning of ImageNet models, adding checkpoint scope parameter.	9 days ago
tf_convert_data.py	UPDATE: Pascal VOC implementation: convert to TFRecords.	2 months ago
tf_utils.py	FIX: Fine tuning of ImageNet models, adding checkpoint scope parameter.	9 days ago
train_ssd_network.py	FIX: Fine tuning of ImageNet models, adding checkpoint scope parameter.	9 days ago

README.md

SSD: Single Shot MultiBox Detector in TensorFlow

SSD is an unified framework for object detection with a single network. It has been originally introduced in this research article.

This repository contains a TensorFlow re-implementation of the original Caffe code. At present, it only implements VGG-based SSD networks (with 300 and 512 inputs), but the architecture of the project is modular, and should make easy the implementation and training of other SSD variants (ResNet or Inception based for instance). Present TF checkpoints have been directly converted from SSD Caffe models.

The organisation is inspired by the TF-Slim models repository containing the implementation of popular architectures (ResNet, Inception and VGG). Hence, it is separated in three main parts:

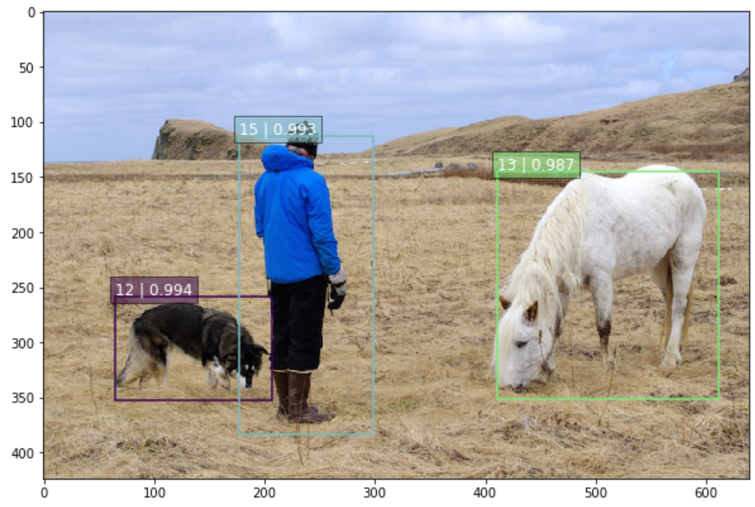
- datasets: interface to popular datasets (Pascal VOC, COCO, ...) and scripts to convert the former to TF-Records;
- networks: definition of SSD networks, and common encoding and decoding methods (we refer to the paper on this precise topic);

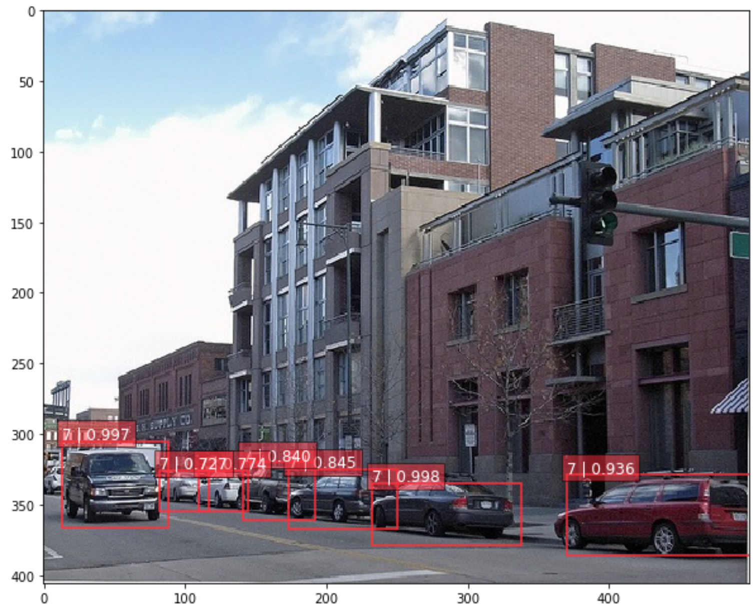
- pre-processing: pre-processing and data augmentation routines, inspired by original VGG and Inception implementations.

SSD minimal example

The SSD Notebook contains a minimal example of the SSD TensorFlow pipeline. Shortly, the detection is made of two main steps: running the SSD network on the image and post-processing the output using common algorithms (top-k filtering and Non-Maximum Suppression algorithm).

Here are two examples of successful detection outputs:





To run the notebook you first have to unzip the checkpoint files in ./checkpoint

```
unzip ssd_300_vgg.ckpt.zip
```

and then start a jupyter notebook with

```
jupyter notebook notebooks/ssd_notebook.ipynb
```

Datasets

The current version only supports Pascal VOC datasets (2007 and 2012). In order to be used for training a SSD model, the former need to be converted to TF-Records using the `tf_convert_data.py` script:

```
DATASET_DIR=./VOC2007/test/
OUTPUT_DIR=./tfrecords
python tf_convert_data.py \
    --dataset_name=pascalvoc \
    --dataset_dir=${DATASET_DIR} \
    --output_name=voc_2007_train \
    --output_dir=${OUTPUT_DIR}
```

Note the previous command generated a collection of TF-Records instead of a single file in order to ease shuffling during training.

Evaluation on Pascal VOC 2007

The present TensorFlow implementation of SSD models have the following performances:

Model	Training data	Testing data	mAP	FPS
SSD-300 VGG-based	VOC07+12 trainval	VOC07 test	0.778	-
SSD-300 VGG-based	VOC07+12+COCO trainval	VOC07 test	0.817	-
SSD-512 VGG-based	VOC07+12+COCO trainval	VOC07 test	0.837	-

We are working hard at reproducing the same performance as the original Caffe implementation!

After downloading and extracting the previous checkpoints, the evaluation metrics should be reproducible by running the following command:

```
EVAL_DIR=./logs/
CHECKPOINT_PATH=./checkpoints/VGG_VOC0712_SSD_300x300_ft_iter_120000.ckpt
python eval_ssd_network.py \
    --eval_dir=${EVAL_DIR} \
    --dataset_dir=${DATASET_DIR} \
    --dataset_name=pascalvoc_2007 \
    --dataset_split_name=test \
    --model_name=ssd_300_vgg \
    --checkpoint_path=${CHECKPOINT_PATH} \
    --batch_size=1
```

The evaluation script provides estimates on the recall-precision curve and compute the mAP metrics following the Pascal VOC 2007 and 2012 guidelines.

In addition, if one wants to experiment/test a different Caffe SSD checkpoint, the former can be converted to TensorFlow checkpoints as following:

```
CAFFE_MODEL=./ckpts/SSD_300x300_ft_VOC0712/VGG_VOC0712_SSD_300x300_ft_iter_120000.caffemodel
python caffe_to_tensorflow.py \
    --model_name=ssd_300_vgg \
    --num_classes=21 \
    --caffemodel_path=${CAFFE_MODEL}
```

Training

The script `train_ssd_network.py` is in charged of training the network. Similarly to TF-Slim models, one can pass numerous options to the training process (dataset, optimiser, hyper-parameters, model, ...). In particular, it is possible to provide a checkpoint file which can be use as starting point in order to fine-tune a network.

Fine-tuning existing SSD checkpoints

The easiest way to fine the SSD model is to use as pre-trained SSD network (VGG-300 or VGG-512). For instance, one can fine a model starting from the former as following:

```
DATASET_DIR=./tfrecords
TRAIN_DIR=./logs/
CHECKPOINT_PATH=./checkpoints/ssd_300_vgg.ckpt
```

```
python train_ssd_network.py \
    --train_dir=${TRAIN_DIR} \
    --dataset_dir=${DATASET_DIR} \
    --dataset_name=pascalvoc_2012 \
    --dataset_split_name=train \
    --model_name=ssd_300_vgg \
    --checkpoint_path=${CHECKPOINT_PATH} \
    --save_summaries_secs=60 \
    --save_interval_secs=600 \
    --weight_decay=0.0005 \
    --optimizer=adam \
    --learning_rate=0.001 \
    --batch_size=32
```

Note that in addition to the training script flags, one may also want to experiment with data augmentation parameters (random cropping, resolution, ...) in `ssd_vgg_preprocessing.py` or/and network parameters (feature layers, anchors boxes, ...) in `ssd_vgg_300/512.py`

Furthermore, the training script can be combined with the evaluation routine in order to monitor the performance of saved checkpoints on a validation dataset. For that purpose, one can pass to training and validation scripts a GPU memory upper limit such that both can run in parallel on the same device. If some GPU memory is available for the evaluation script, the former can be run in parallel as follows:

```
EVAL_DIR=${TRAIN_DIR}/eval
python eval_ssd_network.py \
    --eval_dir=${EVAL_DIR} \
    --dataset_dir=${DATASET_DIR} \
    --dataset_name=pascalvoc_2007 \
    --dataset_split_name=test \
    --model_name=ssd_300_vgg \
    --checkpoint_path=${TRAIN_DIR} \
    --wait_for_checkpoints=True \
    --batch_size=1 \
    --max_num_batches=500
```

Fine-tuning a network trained on ImageNet

One can also try to build a new SSD model based on standard architecture (VGG, ResNet, Inception, ...) and set up on top of it the `multibox` layers (with specific anchors, ratios, ...). For that purpose, you can fine-tune a network by only loading the weights of the original architecture, and initialize randomly the rest of network. For instance, in the case of the VGG-16 architecture, one can train a new model as following:

```
DATASET_DIR=./tfrecords
TRAIN_DIR=./logs/
CHECKPOINT_PATH=./checkpoints/vgg_16.ckpt
python train_ssd_network.py \
    --train_dir=${TRAIN_DIR} \
    --dataset_dir=${DATASET_DIR} \
    --dataset_name=pascalvoc_2007 \
    --dataset_split_name=train \
    --model_name=ssd_300_vgg \
    --checkpoint_path=${CHECKPOINT_PATH} \
    --checkpoint_model_scope=vgg_16 \
    --checkpoint_exclude_scopes=ssd_300_vgg/conv6,ssd_300_vgg/conv7,ssd_300_vgg/block8,ssd_300_vgg/block9,s
    --save_summaries_secs=60 \
    --save_interval_secs=600 \
    --weight_decay=0.0005 \
    --optimizer=rmsprop \
    --learning_rate=0.0001 \
    --batch_size=32
```

Hence, in the former command, the training script randomly initializes the weights belonging to the `checkpoint_exclude_scopes` and load from the checkpoint file `vgg_16.ckpt` the remaining part of the network.

Note that a number of pre-trained weights of popular architectures can be found on TF-Slim models page.