Faster RCNN的一个Tensorflow实现

Faster RCNN的一个Tensorflow实现

tf-faster-rcnn

A Tensorflow implementation of faster RCNN detection framework by Xinlei Chen (xinleic@cs.cmu.edu). This repository is based on the python Caffe implementation of faster RCNN available here.

Note: Several minor modifications are made when reimplementing the framework, which gives potential improvements. For details about the modifications and ablative analysis, please refer to the technical report An Implementation of Faster RCNN with Study for Region Sampling. If you are seeking to reproduce the results in the original paper, please use the official code or maybe the semi-official code. For details about the faster RCNN architecture please refer to the paper Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.

Detection Performance

We only tested it on plain VGG16 architecture so far. Our best performance as of Feburary 2017 (single model on `conv5_3`, no multi-scale, no multi-stage bounding box regression, no skip-connection, no extra input):

- Train on VOC 2007 trainval and test on VOC 2007 test, **71.2**.

热门度与活跃度
1.3 10.0

最新发布的版本
版本：v0.12
Source code(tar.gz)
Source code(zip)
发布时间:2月前

访问主页
访问GitHub主页
Watchers：42
Star：425
Fork：176
创建时间: 2017-01-24 03:51:32
最后Commits： 前天

- Train on COCO 2014 trainval-minival and test on minival (longer), **29.1**.

Note that:

- The above numbers are obtained with a different testing scheme without selecting region proposals using non-maximal suppression (TEST.MODE top), the default and original testing scheme (TEST.MODE nms) will result in slightly worse performance (see report, for COCO it drops 0.3 - 0.4 AP).
- Since we keep the small proposals (< 16 pixels width/height), our performance is especially good for small objects.
- For coco, we find the performance improving with more iterations (600k/790k: 28.3, 800k/109k: 29.1), and potentially better performance can be achieved with even more iterations. Check out here for the latest model.

Additional Features

Additional features not mentioned in the report are added to make research life easier:

- **Support for train-and-validation**. During training, the validation data will also be tested from time to time to monitor the process and check potential overfitting. Ideally training and validation should be separate, where the model is loaded everytime to test on validation. However I have implemented it in a joint way to save time and GPU memory. Though in the default setup the testing data is used for validation, no special attempts is made to overfit on testing set.
- **Support for resuming training**. I tried to store as much information as possible when snapshoting, with the purpose to resume training from the lateset snapshot properly. The meta information includes current image index, permutation of

endernewton
@endernewton

查看开发者资料

文章目录

基本信息

分类：机器学习

收录时间：2017-02-11 09:19:35

images, and random state of numpy. However, when you resume training the random seed for tensorflow will be reset (not sure how to save the random state of tensorflow now), so it will result in a difference. **Note** that, the current implementation still cannot force the model to behave deterministically even with the random seeds set. Suggestion/solution is welcome and much appreciated.
- **Support for visualization**. The current implementation will summarize statistics of losses, activations and variables during training, and dump it to a separate folder for tensorboard visualization. The computing graph is also saved for debugging.

Prerequisites

- A basic Tensorflow installation. r0.12 is fully tested. r0.10+ should in general be fine. While it is not required, for experimenting the original RoI pooling (which requires modification of the C++ code in tensorflow), you can check out my tensorflow fork and look for `tf.image.roi_pooling` .
- Python packages you might not have: `cython` , `python-opencv` , `easydict` (similar to py-faster-rcnn).

Installation

1. Clone the repository

```
git clone https://github.com/endernewton/tf-faster-rcn
```

2. Build the Cython modules

```
cd tf-faster-rcnn/lib
make
```

3. Download pre-trained models and weights

```
# return to the repository root
cd ..
# model for both voc and coco using default training
./data/scripts/fetch_faster_rcnn_models.sh
# model for coco using longer training scheme (600k/7
./data/scripts/fetch_coco_long_models.sh
# weights for imagenet pretrained model, extracted fr
./data/scripts/fetch_imagenet_weights.sh
```

Right now the imagenet weights are used to initialize layers for both training and testing to build the graph, despite that for testing it will later restore trained tensorflow models. This step can be removed in a simplified version.

Setup data

Please follow the instructions of py-faster-rcnn here to setup VOC and COCO datasets. The steps involve downloading data and creating softlinks in the `data` folder. Since faster RCNN does not rely on pre-computed proposals, it is safe to ignore the steps that setup proposals.

If you find it useful, the `data/cache` folder created on my side is also shared here.

Testing

1. Create a folder and a softlink to use the pretrained model

```
mkdir -p output/vgg16/
ln -s data/faster_rcnn_models/voc_2007_trainval/ outpu
ln -s data/faster_rcnn_models/coco_2014_train+coco_201
```

2. Test

```
GPU_ID=0
./experiments/scripts/test_vgg16.sh $GPU_ID pascal_voc
./experiments/scripts/test_vgg16.sh $GPU_ID coco
```

It generally needs several GBs to test the pretrained model (4G on my side).

Training

1. (Optional) If you have just tested the model, first remove the link to the pretrained model

```
rm -v output/vgg16/voc_2007_trainval
rm -v output/vgg16/coco_2014_train+coco_2014_valminusm
```

2. Train (and test, evaluation)

```
GPU_ID=0
./experiments/scripts/vgg16.sh $GPU_ID pascal_voc
./experiments/scripts/vgg16.sh $GPU_ID coco
```

3. Visualization with Tensorboard

```
tensorboard --logdir=tensorboard/vgg16/voc_2007_trainv
tensorboard --logdir=tensorboard/vgg16/coco_2014_train
```

By default, trained networks are saved under:

```
output/<network name>/<dataset name>/default/
```

Test outputs are saved under:

```
output/<network name>/<dataset name>/default/<network snap
```

Tensorboard information for train and validation is saved under:

```
tensorboard/<network name>/<dataset name>/default/
tensorboard/<network name>/<dataset name>/default_val/
```

The default number of training iterations is kept the same to the original faster RCNN, however I find it is beneficial to train longer for COCO (see report). Also note that due to the nondeterministic nature of the current implementation, the performance can vary a bit, but in general it should be within 1% of the reported numbers. Solutions are welcome.

Citation

If you find this implementation or the analysis conducted in our report helpful, please consider citing:

```
@article{chen17implementation,
    Author = {Xinlei Chen and Abhinav Gupta},
    Title = {An Implementation of Faster RCNN with Study
    Journal = {arXiv preprint arXiv:1702.02138},
    Year = {2017}
}
```

For convenience, here is the faster RCNN citation:

```
@inproceedings{renNIPS15fasterrcnn,
    Author = {Shaoqing Ren and Kaiming He and Ross Girsh
    Title = {Faster {R-CNN}: Towards Real-Time Object Det
             with Region Proposal Networks},
    Booktitle = {Advances in Neural Information Processin
    Year = {2015}
}
```