Detection: Faster R-CNN

2016-09-22

Summary

This post records my experience with py-faster-rcnn, including how to setup py-faster-rcnn from scratch, how to perform a demo training on PASCAL VOC dataset by py-faster-rcnn, how to train your own dataset, and some errors I encountered. All the steps are based on Ubuntu 14.04 + CUDA 8.0. Faster R-CNN is an important research result for object detection with an end-to-end deep convolutional neural network architure. For the details, please refer to original paper or my summary.

The source code is provided at: https://github.com/Huangying-Zhan/py-faster-rcnn

Contents

Part 1. Setup py-faster-rcnn

In this part, a simple instruction for install py-faster-rcnn is introduced. The instruction mainly refers to py-faster-rcnn.

1. Clone the Faster R-CNN repo

   ```
   # Make sure to clone with --recursive
   $ git clone --recursive https://github.com/rbgirshick/py-faster-rcnn.git
   ```

2. Lets call the directory as `$FRCN`

3. Build the Cython modules

   ```
   $ cd $FRCN/lib
   $ make
   ```

4. Build Caffe and PyCaffe

   For this part, please refer to Caffe official installation instruction or my post about Caffe installation. If you have experience with Caffe, just follow the

instruction below.

```
$ cd $FRCN/caffe-fast-rcnn
$ cp Makefile.config.example Makefile.config

# Modify Makefile.config, uncommment this line
WITH_PYTHON_LAYER := 1
# Modifiy Makefile.config according to your need, such as setup related to
 GPU support, cuDNN, CUDA version, Anaconda, OpenCV, etc.

# After modification on Makefile.config
$ make all -j4 # -j4 is for complilation acceleration only. 4 is the numbe
r of core in your CPU, change it according to your computer CPU.
# Suppose you have installed prerequites for PyCaffe, otherwise, go back t
o the Caffe installation instructions.
$ make pycaffe -j4
```

5. Download pre-computed Faster R-CNN models

```
$ cd $FRCN
$ ./data/scripts/fetch_faster_rcnn_models.sh
```

6. Run the demo

   However, in this part you might get into trouble with different errors, such as without some packages. At the end of this post, some encountered errors and solution are provided. For those unexpected error, google the error and you should be able to find a solution.

```
$ ./tools/demo.py
```

Part 2. Demo Training on PASCAL VOC

In this part, the training of py-faster-rcnn will be explained. Firstly, an original training procedure on PASCAL VOC dataset is provided. The purpose is to understand the structure of dataset and training steps.

2.1. Prepare dataset and Pre-trained model

1. Download VOC dataset

```
$ cd $FRCN/data
$ wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2
007.tar
$ wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtest_06-Nov-2007.
tar
$ wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCdevkit_08-Jun-200
7.tar
```

```
$ tar xvf VOCdevkit_08-Jun-2007.tar
$ tar xvf VOCtrainval_06-Nov-2007.tar
$ tar xvf VOCtest_06-Nov-2007.tar

$ ln -s VOCdevkit VOCdevkit2007 #create a softlink
```

2. Download pre-trained models

```
$ cd $FRCN
$ ./data/scripts/fetch_imagenet_models.sh
$ ./data/scripts/fetch_faster_rcnn_models.sh
```

2.2. Training

There are 2 types of training methods provided by py-faster-rcnn. One is using the alternating optimization algrithm while another one is approximate joint training method. In this post, approximate joint training method is introduced. For the details, please refer to the paper, Faster R-CNN.

```
$ cd $FRCN
# ./experiments/scripts/faster_rcnn_end2end.sh [GPU_ID] [NET] [DATASET]
# Directly run this command might have an error "AssertionError: Selective searc
h data not found at:". For the solution, please refer to Part 4.
$ ./experiments/scripts/faster_rcnn_end2end.sh 0 ZF pascal_voc
```

Here is a remark about the logic and idea behind the training script.

1. `faster_rcnn_end2end.sh`

   This is a shell script, which is the toppest layer of the whole pipeline, it monitors the input arguments, including GPU ID, network structure(ZF-Net, VGG, or others), dataset (PASCAL VOC, COCO or others), and extra configurations.

   ```
   # Part of the script
   GPU_ID=$1
   NET=$2
   NET_lc=${NET,,}
   DATASET=$3

   array=( $@ )
   len=${#array[@]}
   EXTRA_ARGS=${array[@]:3:$len}
   EXTRA_ARGS_SLUG=${EXTRA_ARGS// /_}
   ```

   Then, it will call two programs, `train_net.py` and followed by `test_net.py`. As the name given, `train_net.py` is to train a model while `test_net.py` is to evaluate performance of the trained model.

```
# Part of the script
time ./tools/train_net.py --gpu ${GPU_ID} \
  --solver models/${PT_DIR}/${NET}/faster_rcnn_end2end/solver.prototxt \
  --weights data/imagenet_models/${NET}.v2.caffemodel \
  --imdb ${TRAIN_IMDB} \
  --iters ${ITERS} \
  --cfg experiments/cfgs/faster_rcnn_end2end.yml \
  ${EXTRA_ARGS}

set +x
NET_FINAL=`grep -B 1 "done solving" ${LOG} | grep "Wrote snapshot" | awk
 '{print $4}'`
set -x

time ./tools/test_net.py --gpu ${GPU_ID} \
  --def models/${PT_DIR}/${NET}/faster_rcnn_end2end/test.prototxt \
  --net ${NET_FINAL} \
  --imdb ${TEST_IMDB} \
  --cfg experiments/cfgs/faster_rcnn_end2end.yml \
  ${EXTRA_ARGS}
```

2. `faster_rcnn_end2end.yml`

   As we can see from `faster_rcnn_end2end.sh`, `cfg` comes from `faster_rcnn_end2end.yml`, which means that this file stores many importatnt configurations. Here shows some original configurations provided.

```
EXP_DIR: faster_rcnn_end2end
TRAIN:
  HAS_RPN: True
  IMS_PER_BATCH: 1
  BBOX_NORMALIZE_TARGETS_PRECOMPUTED: True
  RPN_POSITIVE_OVERLAP: 0.7
  RPN_BATCHSIZE: 256
  PROPOSAL_METHOD: gt
  BG_THRESH_LO: 0.0
TEST:
  HAS_RPN: True
```

   However, if you wish to add your own configurations, such as number of iterations to take a model snapshot while training, you may refer to `$FRCN/lib/fast_rcnn/config.py`. This file contains all the configuration parameters. You don't need to set the configuration in this `config.py` but just add a statement in `faster_rcnn_end2end.yml`. The program can parse the arguments automatically. Of course there exists default values if you do not declare the items in the .yml file.

```
# Example to add SNAPSHOT_ITERS into the configuration
EXP_DIR: faster_rcnn_end2end
TRAIN:
  HAS_RPN: True
```

```
  IMS_PER_BATCH: 1
  BBOX_NORMALIZE_TARGETS_PRECOMPUTED: True
  RPN_POSITIVE_OVERLAP: 0.7
  RPN_BATCHSIZE: 256
  PROPOSAL_METHOD: gt
  BG_THRESH_LO: 0.0
  SNAPSHOT_ITERS: 10000 # This line is an example to add arguments.
TEST:
  HAS_RPN: True
```

3. `train_net.py`

   Basically, there are 3 things included in the file.

```
# Read dataset
imdb, roidb = combined_roidb(args.imdb_name)

# Pass configurations from `faster_rcnn_end2end.sh` and `faster_rcnn_end2e
nd.yml` to lower layer programs/functions
# Call `fast_rcnn.train_net` for training
    train_net(args.solver, roidb, output_dir, pretrained_model=args.pretra
ined_model, max_iters=args.max_iters)
```

4. `combined_roidb` & `pascal_voc.py`

   Recall that in `faster_rcnn_end2end.sh`. You have entered an argument,

```
--imdb ${TRAIN_IMDB}
```

   `combined_roidb` do nothing but just trace back and read the datasets, such as train, val, and test using functions in `$FRCN/lib/datasets/pascal_voc.py`.

5. `fast_rcnn.train_net`

   This function is at `$FRCN/lib/fast_rcnn/train.py` This function is the core of whole training pipeline since it calls `solver.prototxt`, but in fact you don't need to care this part in most of the time.

6. `solver.prototxt` & `train.prototxt`

   If you are familiar with Caffe, you should know the purpose of `solver.prototxt` and `train.prototxt`. Otherwise, you are suggested to go through Caffe's MNIST tutorial. In here, the idea will be described briefly only.

   Basically, `solver.prototxt` tells the program where to find your ConvNet structure prototxt and some training setups, such as learning rate, learning policy, etc.

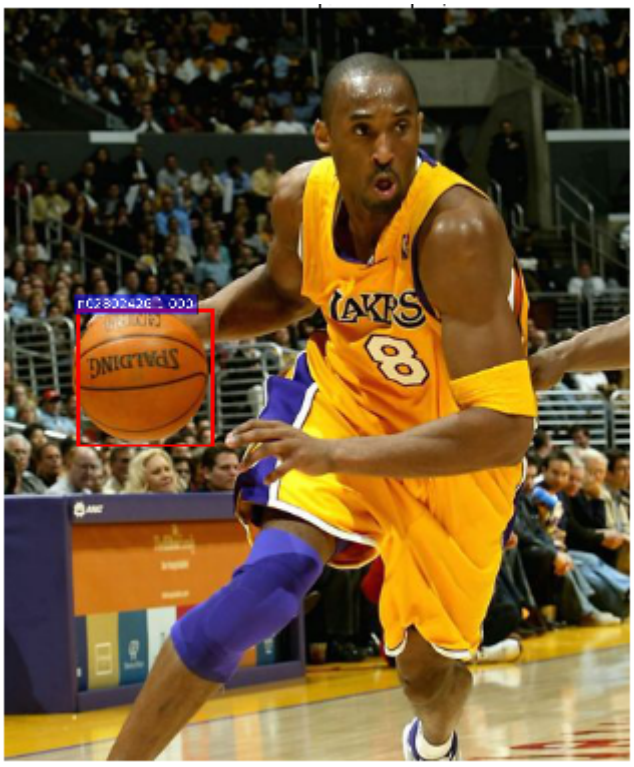
```
train_net: "models/pascal_voc/ZF/faster_rcnn_end2end/train.prototxt"
base_lr: 0.001
```

```
lr_policy: "step"
gamma: 0.1
stepsize: 50000
display: 20
average_loss: 100
momentum: 0.9
weight_decay: 0.0005
snapshot_prefix: "zf_faster_rcnn"
iter_size: 2
```

`train.prototxt` describes the network structure, including number of layer, type of layer, number of neurons in each layer, etc. Again, refer to Caffe's MNIST tutorial in order to understand `train.prototxt`.

Part 3. Training on new dataset

In this part, **basketball detection** will be used as an example to illustrate how to train a new dataset using py-faster-rcnn.



3.1. Prepare dataset

The dataset used in this part is downloaded from ImageNet.

DISCLAIMER: This dataset should be only used for non-commercial research activities. Please follow the ImageNet rules about the use of the dataset.

1. Download dataset

Here provides a link to download Basketball Dataset. This dataset has the following structure.

```
|-- basketball
    |-- JPEGImages
        Contains all raw .JPEG images
    |-- ImageSets
        .txt files state training set, validataion set. Extension is not r
equired in these files
    |-- Annotations
        Bounding boxes annotation for each image. The annotation files are
 written in .xml format.
```

```
# Unzip the folder
$ mv basketball.tar.gz $FRCN/data/
$ cd $FRCN/data
$ tar xzf basketball.tar.gz
```

2. Add a dataset python file

   Add a `basketball.py` to `$FRCN/lib/datasets/` . You may check on the source code for reference. If you wish to modify this file, basically, you can just *find and replace*basketball by your new dataset name.

3. Add `basketball_eval.py`

   Add a `basketball_eval.py` to `$FRCN/lib/datasets/` . Again, check on the source code for reference. Again, *find and replace* basketabll.

4. Update `/lib/datasets/factory.py`

   The purpose of `basketball.py` is to read a part of whole dataset, such as train*set or val*set. The purpose of function in `factory.py` is to get all sets of whole dataset.

5. Add config file

   As mentioned in Part 2.2 (2), we need a `config.yml` to store configurations. In here, we can use the original `faster_rcnn_end2end.yml` as a reference. However, there are many configurations you can set in this file. In here, we may set `EXP_DIR` first and others if necessary.

   ```
   $ cd $FRCN/experiments/cfgs
   $ cp faster_rcnn_end2end.yml config.yml
   ```

6. Update `imdb.py`

   Since new dataset may have conflicts in annotation with original PASCAL VOC dataset. For example, ImageNet images start with index 0 in row and

col while PASCAL VOC dataset starts with index 1. In `imdb.py`, a part of code should be inserted in `append_flipped_images()`. Refer to source code.

3.2. Prepare network and pre-trained model

To train our own model, basically we don't need to train the model from scratch unless you have a huge dataset which is comparable to ImageNet. Otherwise, we can train our model from fine-tuning a pre-trained Faster R-CNN model. The reason is because a pre-trained Faster R-CNN contains a lot of good lower level features, which can be used generally. Even you are using a new and self-defined architure (i.e. no existing pre-trained Faster R-CNN model), follow the training method of Faster R-CNN and train a Faster R-CNN first, followed by fine-tuning on your own dataset is suggested.

For simplicity, the network and model adopted in this part is ZF-net and a pre-trained Faster R-CNN (ZF) respectively.

```
$ cd $FRCN/models
# copy a well-defined network and make modification based on it
$ mkdir basketball
$ cp ./pascal_voc/ZF/faster_rcnn_end2end/* ./basketball/
$ cd basketball
```

Now, we should modify all files in `basketball/`, including,

- `solver.prototxt`
  - train_net
  - snapshot_prefix
  - Others if necessary
- `train.prototxt` & `test.prototxt`

  For `train.prototxt` and `val.prototxt`, basically we need to update the number of output in final layers. Let's say, in this basketball dataset, we only need 2 classes (background + basketball) and 8 output for bounding box regressor. Orignial pascal_voc have 21 classes including background and 21*4 bounding box regressor output.

  ```
  $ cd $FRCN/models/basketball
  $ grep 21 *
  $ grep 84 *
  # These two commands help you to check the lines that you should modify in
   the files.
  ```

In this part, there are two more items we need to modify. Since we are **fine-tuning a pre-trained ConvNet model**on our own dataset and the number of output at last fully-connected layers (cls*score & bbox*pred) has been changed, the **original weight** in pre-trained ConvNet model is not suitable for our current network. The dimension is totally different. The details can be refered to Caffe's fine-tuning tutorial. The solution is to rename the layers such that the weights for the layers will be initialized randomly instead of copying from pre-trained model (actually copying from pre-trained model will cause error).

```
name: "cls_score" -> name: "cls_score_basketball"
name: "bbox_pred" -> name: "bbox_pred_basketball"
```

However, renaming the layers may cause problems in later parts since "cls*score" and "bbox*pred" are used as keys in testing. Therefore, in the training part, we can train the model accroding to the following procedure.

1. Rename the layers to *clsscorebasketball* and *bboxpredbasketball*
2. Fine-tune pre-trained Faster R-CNN (FRCN) model and snapshot at iteration 0. Let's call the snapshot **Basketball_0.caffemodel**. Stop training.
3. Rename the layers back to *cls_score* and *bbox_pred*.
4. Fine-tune **Basketball_0.caffemodel** to get our final model.

The details and code will be explained in the following part.

3.3. Training and evaluation

Before training on your new dataset, you may need to check `$FRCN/data/cache` to remove caches if necessary. Caches stores information of previously trained dataset. It may cause problem while training.

1. Rename the layers

   As mentions in the previous part, rename the two layers.

   Reminder: if you are using *find and replace*, please find the name with quotes(i.e. "cls*score"). If you just search for cls*score, without quotes, it may also replace some other layers since there is a layer named rpn*cls*score.

2. First fine-tuning

   The purpose of first fine-tuning is to get a caffemodel which has two outputs at final fully-connected layers.

```
$ ./tools/train_net.py --gpu 0 --weights data/faster_rcnn_models/ZF_faster
_rcnn_final.caffemodel --imdb basketball_train --cfg experiments/cfgs/conf
ig.yml --solver models/basketball/solver.prototxt --iter 0
```

After this fine-tuning, we should get the model we needed.

3. Rename the layers back

   Rename the two layers back to *"cls_score"* and *"bbox_pred"*.

4. Second fine-tuning

   This fine-tuning should train models for our final use. The pre-trained model in this stage is the model we saved in stage 2.

```
$ ./tools/train_net.py --gpu 0 --weights output/basketball/train/zf_faster
_rcnn_basketball_iter_0.caffemodel --imdb basketball_train --cfg experimen
ts/cfgs/config.yml --solver models/basketball/solver.prototxt --iter 10000
```

5. Evaluation / Testing

   To test the performance of trained model, we can use the provided `test_net.py` for the purpose.

```
$ ./tools/test_net.py --gpu 0 --def models/basketball/test.prototxt --net
 output/basketball/train/zf_faster_rcnn_basketball_iter_20000.caffemodel -
-imdb basketball_val --cfg experiments/cfgs/config.yml
```

   At the end, you should be able to see something like this.

```
im_detect: 174/174 0.078s 0.001s
Evaluating detections
Writing n02802426 basketball results file
AP for n02802426 = 0.8966
Mean AP = 0.8966
~~~~~~~~
Results:
0.897
0.897
~~~~~~~~

--------------------------------------------------------------
Results computed with the **unofficial** Python eval code.
Results should be very close to the official MATLAB eval code.
Recompute with `./tools/reval.py --matlab ...` for your paper.
-- Thanks, The Management
--------------------------------------------------------------
```

After going through such long path, training on py-faster-rcnn is completed!

Part 4. Error and solution

1. no easydict, cv2

```
# Without Anaconda
$ sudo pip install easydict
$ sudo apt-get install python-opencv

# With Anaconda
$ conda install -c verydeep easydict
$ conda install opencv
# Normally, people will follow the online instruction at https://anaconda.
org/auto/easydict and install auto/easydict. However, this easydict (ver.
1.4) has a problem in passing the message of configuration and cause many
 unexpected error while verydeep/easydict (ver.1.6) won't cause these erro
rs.
```

2. libcudart.so.8.0: cannot open shared object file: No such file or directory

```
 $ sudo ldconfig /usr/local/cuda/lib64
```

3. assertionError: Selective Search data is not found

   Solution: install verydeep/easydict rather than auto/easydict

```
$ conda install -c verydeep easydict
```

4. box [:, 0] > box[:, 2]

   Solution: add the following code block in imdb.py

```
def append_flipped_images(self):
    num_images = self.num_images
    widths = self._get_widths()
    for i in xrange(num_images):
        boxes = self.roidb[i]['boxes'].copy()
        oldx1 = boxes[:, 0].copy()
        oldx2 = boxes[:, 2].copy()
        boxes[:, 0] = widths[i] - oldx2
        boxes[:, 2] = widths[i] - oldx1
        for b in range(len(boxes)):
                if boxes[b][2] < boxes[b][0]:
                    boxes[b][0]=0
        assert (boxes[:, 2] >= boxes[:, 0]).all()
```

5. For ImageNet detection dataset, no dneed to minus one on coordinates

```
# Load object bounding boxes into a data frame.
for ix, obj in enumerate(objs):
    bbox = obj.find('bndbox')
    # Make pixel indexes 0-based
    x1 = float(bbox.find('xmin').text)
    y1 = float(bbox.find('ymin').text)
    x2 = float(bbox.find('xmax').text)
    y2 = float(bbox.find('ymax').text)
    cls = self._class_to_ind[obj.find('name').text.lower().strip()]
```

Reference

zhang_shuai's blog

deboc's tutorial