Faster RCNN Documentation

Release 0.1

anhlt

Jan 29, 2019

Contents

CHAPTER 1

Main features

- Faster RCNN for *hooman*
- End to End Training support by default
- Multibatch (images) support
- Multi-GPU support
- Docker images provided
- Tensorboard support
- Good *(not yet)* Documentation

CHAPTER 2

Install

Clone at https://github.com/anhlt/faster_rcnn

2.1 Installation

2.1.1 System requirements

- Nvidia gpu (GTX 10x0 or later, earlier may work but not tested)
- OS: Linux (We prefer Ubuntu 16.04)

We provide the `Dockerfile` and `docker-compose.yml` for you to install the enviroment. First we need to install docker in host machine

2.1.2 Install Docker

1. Install Docker

   Install Docker from Homepage

2. Install CUDA on Host Machine

   Install Cuda 8.0 on Ubuntu 16.04

3. Install docker-compose

   https://docs.docker.com/compose/install/

4. Install Nvidia-docker

   In order to passthrough GPU to docker we need to install Nvidia-docker

2.1.3 Create Docker image

1. Clone from github:

```
git clone git@github.com:anhlt/faster_rcnn.git
```

2. Use docker-compose to create a docker image

```
cd ~/workspace/faster_rcnn
docker-compose up --build
```

2.1.4 Compile Cython module

There are 3 modules need to be compiled, `nms`, `roi_pooling`, `utils`. We need to exec `\bin\bash` on Docker image to build those modules

```
cd ~/workspace/faster_rcnn
docker-compose exec python /bin/bash
```

- Compile `nms`

```
cd /data/faster_rcnn/nms
python setup.py build_ext --inplace
rm -rf build
```

- Compile `utils`

```
cd /data/faster_rcnn/utils
python setup.py build_ext --inplace
```

- Compile `roi_pooling`

```
cd /data/faster_rcnn/roi_pooling/src/cuda/
nvcc -c -o roi_pooling.cu.o roi_pooling_kernel.cu -D GOOGLE_CUDA=1 -x cu -
↪Xcompiler -fPIC -arch=sm_61

cd /data/faster_rcnn/roi_pooling
python setup.py build_ext --inplace
```

CHAPTER 3

Tutorial

This part of the documentation, tutorial to train faster-rcnn on custom datasets

3.1 MSCOCO dataset

```
data
├── annotations
│   ├── captions_train2014.json
│   ├── captions_val2014.json
│   ├── instances_train2014.json
│   ├── instances_val2014.json
│   ├── person_keypoints_train2014.json
│   └── person_keypoints_val2014.json
└── images
    ├── train2014
    └── val2014
```

3.2 VOC dataset

3.2.1 Datasets

Sample flower dataset that contain total 280 images, belong to 2 species. Download

Extract the downloaded file in to `data` forder in project root like this

```
├── data
│   └── VOC2007
│       ├── Annotations
│       ├── ImageSets
│       ├── JPEGImages
│       └── pascal_label_map.pbtxt
```

3.2.2 Training on Jypiter Notebook

Import necessary modules

```
from faster_rcnn.utils.datasets.voc.voc import VOCDetection
import numpy as np
import torch

from faster_rcnn.utils.datasets.data_generator import CocoGenerator
from faster_rcnn.utils.datasets.data_generator import Enqueuer
from torch.optim import SGD

from faster_rcnn.faster_rcnn import FastRCNN, RPN
from pycrayon import CrayonClient
from torch.optim.lr_scheduler import StepLR
from datetime import datetime
from faster_rcnn.utils.datasets.adapter import convert_data
from faster_rcnn.utils.evaluate.metter import AverageMeter
from faster_rcnn.utils.display.images import imshow, result_show
```

Read Train and Validation datasets

```
root = '/data/data'
ds = VOCDetection(root, 'train')
val_ds = VOCDetection(root, 'val')
print(len(ds), len(val_ds))
```

Enqueue 2 datasets

```
batch_size = 3
data_gen = CocoGenerator(data=ds, batch_size=batch_size, shuffle=True)
queue = Enqueuer(generator=data_gen, use_multiprocessing=False)
queue.start(max_queue_size=20, workers=4)
train_data_generator = queue.get()
```

```
val_data_gen = CocoGenerator(data=ds, batch_size=batch_size, shuffle=True, seed=2)
val_queue = Enqueuer(generator=data_gen, use_multiprocessing=False)
val_queue.start(max_queue_size=20, workers=4)
val_data_generator = val_queue.get()
```

Create Faster RCNN Network

```
categories = ds.classes
print(categories)
net = FastRCNN(categories, debug=False)
net.cuda()
```

Select trainable parameters, and choose optimization strategy

```
params = filter(lambda x: x.requires_grad, net.parameters())
optimizer = SGD(params, lr=1e-4, momentum=0.9, weight_decay=0.0005)
exp_lr_scheduler = StepLR(optimizer, step_size=100, gamma=0.8)
```

Define Evaluate Function

```
def evaluate(data_gen ,model, steps_per_epoch, epochs=1):
    exp_name = datetime.now().strftime('vgg16_%m-%d_%H-%M-%s')
    cc = CrayonClient(hostname="crayon", port=8889)
    exp = cc.create_experiment(exp_name)

    model.eval()
    val_loss = AverageMeter()
    val_cross_entropy = AverageMeter()
    val_loss_box = AverageMeter()
    val_rpn_loss = AverageMeter()

    for epoch in range(epochs):
        for step in range(1, steps_per_epoch +1):
            blobs = data_gen.next()
            batch_tensor, im_info, batch_boxes, batch_boxes_index = convert_
↪data(blobs)
            cls_prob, bbox_pred, rois = model(batch_tensor, im_info, batch_boxes,␣
↪batch_boxes_index)

            loss = model.loss

            val_loss_box.update(model.loss_box.item())
            val_cross_entropy.update(model.cross_entropy.item())
            val_loss.update(loss.item())
            val_rpn_loss.update(model.rpn.loss.item())

    log_text = 'val_loss: %.4f' % (val_loss.avg)
    print(log_text)
    return val_loss ,val_cross_entropy, val_loss_box, val_rpn_loss
```

Define Train Function

```
def train(train_data_gen, val_data_gen, optimizer, lr_scheduler ,model, epochs, steps_
↪per_epoch, val_step_per_epoch):

    exp_name = datetime.now().strftime('vgg16_%m-%d_%H-%M-%s')
    cc = CrayonClient(hostname="crayon", port=8889)
    exp = cc.create_experiment(exp_name)


    train_loss = AverageMeter()
    cross_entropy = AverageMeter()
    loss_box = AverageMeter()
    rpn_loss = AverageMeter()
    current_step = 0
```

(continues on next page)

(continued from previous page)

```
    for epoch in range(epochs):
        model.train()
        for step in range(1, steps_per_epoch +1):
            lr_scheduler.step()
            blobs = data_gen.next()
            batch_tensor, im_info, batch_boxes, batch_boxes_index = convert_
↪data(blobs)

            cls_prob, bbox_pred, rois = model(batch_tensor, im_info, batch_boxes,␣
↪batch_boxes_index)
            cls_data = cls_prob.data.cpu().numpy()
            max_class_idx = np.argmax(cls_data, axis=1)
            loss = model.loss
            cross_entropy.update(model.cross_entropy.item())
            loss_box.update(model.loss_box.item())
            train_loss.update(loss.item())
            rpn_loss.update(model.rpn.loss.item())
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            current_step = epoch * steps_per_epoch + step
            if step % 10 == 0:
                log_text = 'epoch: %d : step %d,  loss: %.4f' % (
                    epoch + 1, step , train_loss.avg)
                print(log_text)

                re_cnt = True
            if step % 10 == 0:
                exp.add_scalar_value('train_loss', train_loss.avg, step=current_step)
                exp.add_scalar_value('rpn_loss', rpn_loss.avg, step=current_step)
                exp.add_scalar_value('cross_entropy', cross_entropy.avg, step=current_
↪step)
                exp.add_scalar_value('loss_box', loss_box.avg, step=current_step)



        torch.save(model.state_dict(), './checkpoints/faster_model_at_epoch_%s.pkl' %␣
↪epoch + 1)
        val_loss ,val_cross_entropy, val_loss_box, val_rpn_loss = evaluate(val_data_
↪gen, model, val_step_per_epoch)
        exp.add_scalar_value('val_loss', val_loss.avg, step=current_step)
        exp.add_scalar_value('val_rpn_loss', val_rpn_loss.avg, step=current_step)
        exp.add_scalar_value('val_cross_entropy', val_cross_entropy.avg, step=current_
↪step)
        exp.add_scalar_value('val_loss_box', val_loss_box.avg, step=current_step)
```

Train Network

```
train(train_data_generator, val_data_generator ,optimizer=optimizer,lr_scheduler=exp_
↪lr_scheduler, model=net, epochs=10, steps_per_epoch=100, val_step_per_epoch=15)
```

```


epoch: 1 : step 10,  loss: 5.0192
```

(continues on next page)

(continued from previous page)

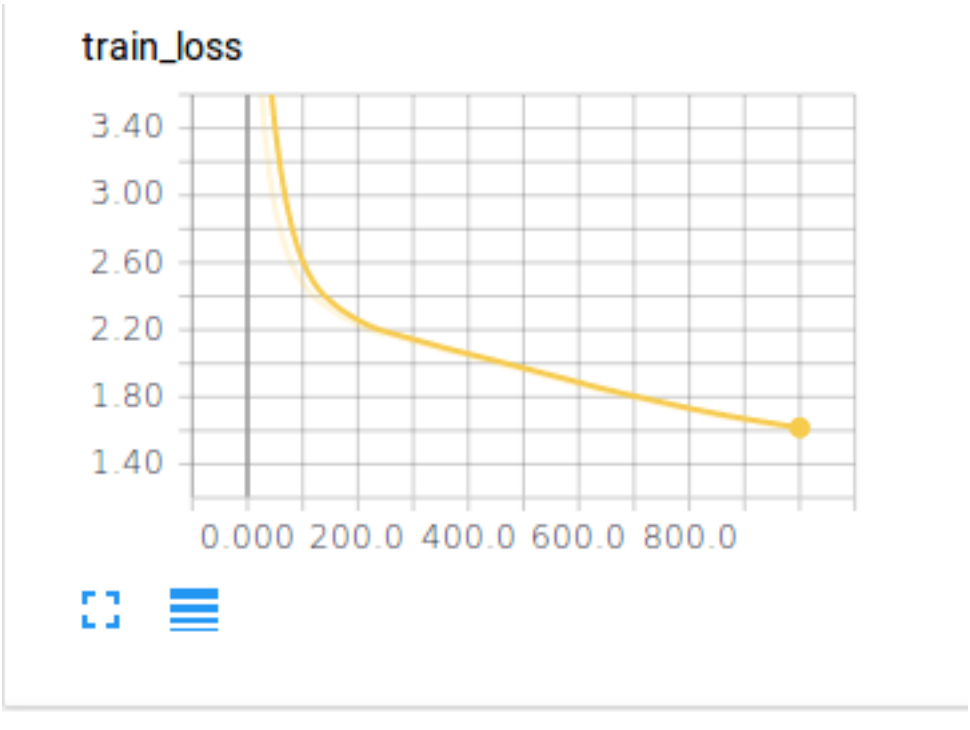
```
epoch: 1 : step 20,  loss: 3.9577
epoch: 1 : step 30,  loss: 3.4115
epoch: 1 : step 40,  loss: 3.1217
epoch: 1 : step 50,  loss: 2.9148
epoch: 1 : step 60,  loss: 2.7841
epoch: 1 : step 70,  loss: 2.6765
epoch: 1 : step 80,  loss: 2.5952
epoch: 1 : step 90,  loss: 2.5265
epoch: 1 : step 100,  loss: 2.4733
val_loss: 2.0767
epoch: 2 : step 10,  loss: 2.4353
epoch: 2 : step 20,  loss: 2.3974
epoch: 2 : step 30,  loss: 2.3686
epoch: 2 : step 40,  loss: 2.3471
epoch: 2 : step 50,  loss: 2.3279
epoch: 2 : step 60,  loss: 2.3041
epoch: 2 : step 70,  loss: 2.2816
epoch: 2 : step 80,  loss: 2.2663
epoch: 2 : step 90,  loss: 2.2476
epoch: 2 : step 100,  loss: 2.2308
val_loss: 1.8522
epoch: 3 : step 10,  loss: 2.2142
epoch: 3 : step 20,  loss: 2.2033
epoch: 3 : step 30,  loss: 2.1916
epoch: 3 : step 40,  loss: 2.1806
epoch: 3 : step 50,  loss: 2.1749
epoch: 3 : step 60,  loss: 2.1665
epoch: 3 : step 70,  loss: 2.1589
epoch: 3 : step 80,  loss: 2.1485
epoch: 3 : step 90,  loss: 2.1395
epoch: 3 : step 100,  loss: 2.1315
val_loss: 1.8585
epoch: 4 : step 10,  loss: 2.1210
epoch: 4 : step 20,  loss: 2.1123
epoch: 4 : step 30,  loss: 2.1034
epoch: 4 : step 40,  loss: 2.0959
epoch: 4 : step 50,  loss: 2.0844
epoch: 4 : step 60,  loss: 2.0772
epoch: 4 : step 70,  loss: 2.0690
epoch: 4 : step 80,  loss: 2.0610
epoch: 4 : step 90,  loss: 2.0543
epoch: 4 : step 100,  loss: 2.0450
val_loss: 1.9057
epoch: 5 : step 10,  loss: 2.0351
epoch: 5 : step 20,  loss: 2.0286
epoch: 5 : step 30,  loss: 2.0205
epoch: 5 : step 40,  loss: 2.0121
epoch: 5 : step 50,  loss: 2.0036
epoch: 5 : step 60,  loss: 1.9942
epoch: 5 : step 70,  loss: 1.9853
epoch: 5 : step 80,  loss: 1.9778
epoch: 5 : step 90,  loss: 1.9683
epoch: 5 : step 100,  loss: 1.9611
val_loss: 1.7018
epoch: 6 : step 10,  loss: 1.9517
epoch: 6 : step 20,  loss: 1.9427
epoch: 6 : step 30,  loss: 1.9341
```

(continues on next page)

(continued from previous page)

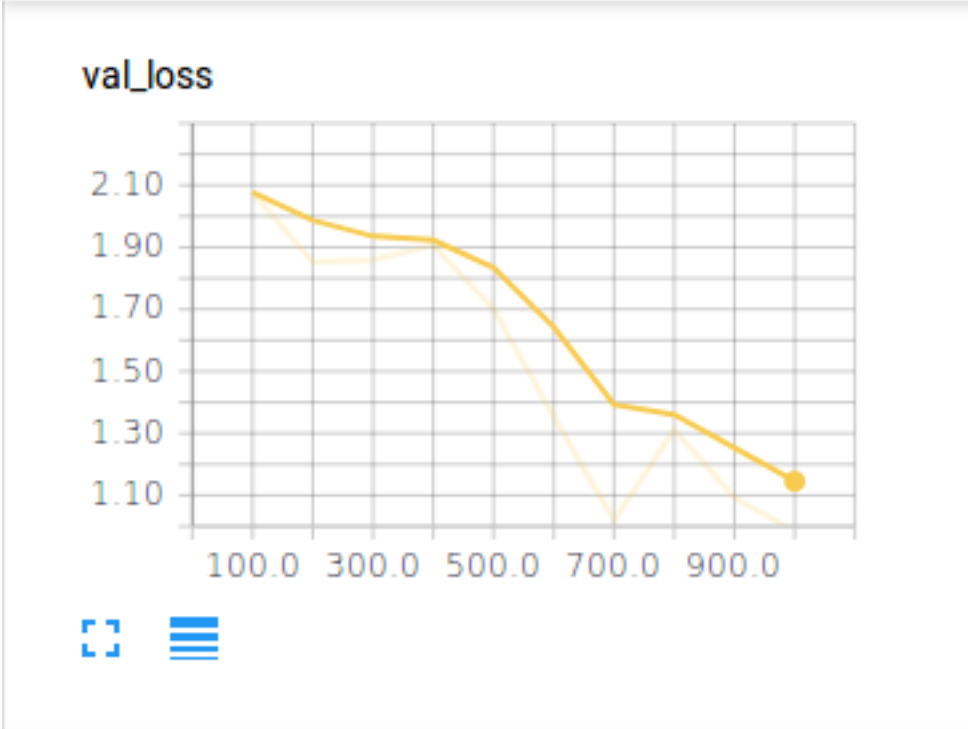
```
epoch: 6 : step 40,  loss: 1.9255
epoch: 6 : step 50,  loss: 1.9177
epoch: 6 : step 60,  loss: 1.9084
epoch: 6 : step 70,  loss: 1.9009
epoch: 6 : step 80,  loss: 1.8926
epoch: 6 : step 90,  loss: 1.8828
epoch: 6 : step 100,  loss: 1.8731
val_loss: 1.3541
epoch: 7 : step 10,  loss: 1.8655
epoch: 7 : step 20,  loss: 1.8566
epoch: 7 : step 30,  loss: 1.8493
epoch: 7 : step 40,  loss: 1.8406
epoch: 7 : step 50,  loss: 1.8329
epoch: 7 : step 60,  loss: 1.8260
epoch: 7 : step 70,  loss: 1.8180
epoch: 7 : step 80,  loss: 1.8088
epoch: 7 : step 90,  loss: 1.8023
epoch: 7 : step 100,  loss: 1.7955
val_loss: 1.0189
epoch: 8 : step 10,  loss: 1.7876
epoch: 8 : step 20,  loss: 1.7805
epoch: 8 : step 30,  loss: 1.7740
epoch: 8 : step 40,  loss: 1.7653
epoch: 8 : step 50,  loss: 1.7580
epoch: 8 : step 60,  loss: 1.7516
epoch: 8 : step 70,  loss: 1.7442
epoch: 8 : step 80,  loss: 1.7368
epoch: 8 : step 90,  loss: 1.7295
epoch: 8 : step 100,  loss: 1.7228
val_loss: 1.3117
epoch: 9 : step 10,  loss: 1.7163
epoch: 9 : step 20,  loss: 1.7110
epoch: 9 : step 30,  loss: 1.7038
epoch: 9 : step 40,  loss: 1.6972
epoch: 9 : step 50,  loss: 1.6908
epoch: 9 : step 60,  loss: 1.6844
epoch: 9 : step 70,  loss: 1.6787
epoch: 9 : step 80,  loss: 1.6738
epoch: 9 : step 90,  loss: 1.6679
epoch: 9 : step 100,  loss: 1.6624
val_loss: 1.0927
epoch: 10 : step 10,  loss: 1.6569
epoch: 10 : step 20,  loss: 1.6523
epoch: 10 : step 30,  loss: 1.6458
epoch: 10 : step 40,  loss: 1.6407
epoch: 10 : step 50,  loss: 1.6361
epoch: 10 : step 60,  loss: 1.6304
epoch: 10 : step 70,  loss: 1.6242
epoch: 10 : step 80,  loss: 1.6189
epoch: 10 : step 90,  loss: 1.6141
epoch: 10 : step 100,  loss: 1.6095
val_loss: 0.9826
```

We also support Tensorboard at http://localhost:8888 you can view your Train loss, and Val loss.



val_cross_entropy

val_loss



3.2.3 Predict

```
pred_boxes, scores, classes, rois, blob = net.detect('./test_im/test5.jpg', thr=0.8)
result_show(blob[0], pred_boxes,classes, scores)
```

You can download the Notebook here

CHAPTER 4

The API Documentation / Guide

If you are looking for information on a specific function, class, or method, this part of the documentation is for you.

4.1 Region Proposal Network

class faster_rcnn.faster_rcnn.**RPN**

Generate region proposals, shares computation with the object detection network.

anchor_scales
list – The scale of each anchor on particular point on feature maps.

anchor_target_layer
faster_rcnn.rpn_msr.anchor_target_layer.AnchorTargerLayer – Calculate network target base on anchors and ground truth boxes.

bbox_conv
torch.nn.module – Proposals coordinate refine predictor

conv1
torch.nn.module – Probability that anchors contains object predictor

cross_entropy
int – Cross entropy loss.

features
torch.nn.module – Backbone network, that share computation with object detection network

loss_box
int – Box coordinate refine loss.

proposal_layer
faster_rcnn.rpn_msr.proposal_layer.ProposalLayer – Create proposals base on generated anchors and bbox refine values.

score_conv
TYPE – Description

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

_computer_forward(*im_data*)
Calculate forward

Parameters **im_data** (torch.tensor) – image as tensor

Returns Return feature map, proposal boxes refine values w.r.t to each anchors, probability that anchors is foreground

Return type (torch.tensor, torch.tensor, torch.tensor)

forward(*im_data*, *im_info*, *gt_boxes=None*, *gt_boxes_index=[]*)
Forward

Parameters

- **im_data** (*TYPE*) – Description
- **im_info** (*TYPE*) – Description
- **gt_boxes** (*None, optional*) – Description
- **gt_boxes_index** (*list, optional*) – Description

Returns Return the features map and list of rois.

Return type tuple(features, rois)

4.2 AnchorTargerLayer

class faster_rcnn.rpn_msr.anchor_target_layer.**AnchorTargerLayer**(*feat_stride*, *anchor_scales*, *is_cuda=True*)
Calculate target for RPN network

is_cuda
bool – Using GPU or not

__init__(*feat_stride*, *anchor_scales*, *is_cuda=True*)
Summary

Parameters

- **feat_stride** (class::*numpy.array*) – The Ratio between original image size and the convolutional feature (feature maps), Used to calculate anchors from a point in convolutional feature into the original image.

  Example: np.array([16. ,])
- **anchor_scales** (class::*numpy.array*) – Description
- **is_cuda** (*bool, optional*) – Description

_create_anchors(*feature_height*, *feature_width*)
Create all anchors given features height, width.

Parameters

- **feature_height** (*int*) – feature map height
- **feature_width** (*int*) – feature map width

Returns Anchors

Return type `numpy.array`

_filter_outside_anchors(*all_anchors*, *im_height*, *im_width*)
Remove outside anchors from generated anchors

Parameters

- **all_anchors** (`numpy.array`) – All generated anchors
- **im_height** (*int*) – Origin image height.
- **im_width** (*int*) – Origin image width

Returns Return all inside anchors and its indexes

Return type tuple(inside_anchors, index_of_inside_anchors)

calculate_target(*inside_anchors*, *batch_size*, *inside_anchor_indexes*, *batch_boxes*, *batch_boxes_index*)
Calculate bbox_targets and layer

Notes

Create empty label array.
- *label*:

```
>>> label.shape
(A, batch_size)
```

For each batch, there are *A* anchors and *G* batch boxes:

current_batch_overlaps: overlaps between anchors and boxes

```
>>> current_batch_overlaps.shape
(A, G)
```

argmax_overlaps : List index of boxes, that have largest overlap w.r.t each Anchor.

```
>>> argmax_overlaps.shape
(A, 1)
```

max_overlaps : List of largest overlap values between boxes w.r.t each Anchor.

```
>>> max_overlaps.shape
(A, 1)
```

Set current batch *label* values:

```
>>> labels[i, max_overlaps < cfg.TRAIN.RPN_NEGATIVE_OVERLAP] = 0
>>> labels[i, max_overlaps >= cfg.TRAIN.RPN_POSITIVE_OVERLAP] = 1
```

Parameters

- **inside_anchors** (numpy.array) – List all anchors that lay inside the original images. Shape: [number_of_anchor * 4]
- **batch_size** (*int*) – Current batch size
- **inside_anchor_indexes** (numpy.array) – Indexes of inside anchors. Shape: [number_of_anchor * 1]
- **batch_boxes** (numpy.array) – list all ground truth boxes across all the images in batch example:

```
>>> batch_boxes
[[  48.57142857  465.71428571  537.14285714 1774.28571429]
 [ 220.         1065.71428571  382.85714286 1771.42857143]
 [ 326.76056338  315.49295775  394.36619718  580.28169014]
 [  76.05633803  290.14084507  242.25352113  788.73239437]
 [  11.26760563    8.45070423  585.91549296 1769.01408451]
 [ 178.125       221.875       287.5         975.        ]
 [ 321.875       334.375       434.375       984.375     ]]
```

- **batch_boxes_index** (*list[Int]*) –

  Batch index where image belong to.

  Example:

  There 3 images in current batch, and 6 boxes inside that 3 images. Look at the *batch_boxes_index* we know:

  First 3 boxes belong to first image.

  Next 2 boxes belong to second image.

  Last box belong to last image.

```
>>> [0, 0, 0, 1, 1, 2]
```

Returns Return caculated labels , and bbox_targers

Return type (numpy.array((A, batch_size)), numpy.array((batch_size, A, 4)))

forward(*rpn_cls_score*, *gt_boxes*, *batch_boxes_index*, *im_info*)

Generate all anchors, then filter anchors that lays outside original images. Calculate target base on overlaps values and bbox regression.

Parameters

- **rpn_cls_score** (torch.Tensor) – The probability of each anchor contains object center
- **gt_boxes** (numpy.array) – List all ground truth boxes across all the images in batch
- **batch_boxes_index** (numpy.array) – Batch index where image belong to.

- **im_info** (torch.Tensor([[im_height, im_width]])) – Original Image size

Returns Return labels, bbox_targets, bbox_inside_weights, bbox_outside_weights

Return type (torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor)

4.3 ProposalLayer

class faster_rcnn.rpn_msr.proposal_layer.**ProposalLayer**(*_feat_stride=[16], anchor_scales=[8, 16, 32]*)

__init__(*_feat_stride=[16], anchor_scales=[8, 16, 32]*)
x.__init__(...) initializes x; see help(type(x)) for signature

_filter_boxes(*boxes*, *min_size*)
Remove all boxes with any side smaller than min_size.

forward(*scores*, *bbox_deltas*, *im_info*, *cfg_key*)
Summary

Notes

for each (H, W) location i
generate A anchor boxes centered on cell i
apply predicted bbox deltas at cell i to each of the A anchors
clip predicted boxes to image
remove predicted boxes with either height or width < threshold
sort all (proposal, score) pairs by score from highest to lowest
take top pre_nms_topN proposals before NMS
apply NMS with threshold 0.7 to remaining proposals
take after_nms_topN proposals after NMS
return the top proposals (-> RoIs top, scores top)

Parameters

- **scores** (*TYPE*) – Description
- **bbox_deltas** (*TYPE*) – Description
- **im_info** (*TYPE*) – Description
- **cfg_key** (*TYPE*) – Description

Returns Description

Return type TYPE

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

B

C

F

I

L

P

R

S