Coding Project 2: Object Detection - Youshaa Murhij

yosha.morheg@gmail.com

Before Coding Let's talk a bit about the paper:

The paper presents a region proposal network (RPN) which shares the conv features with R-CNN and includes two fully connected layers that predicts object bounds and objectness scores at each position. RPN is heuristically combined with Fast R-CNN for object detection and achieves state-of-the-art performance on VOC07 and VOC12. The running time of generating region proposals is greatly reduced since RPN shares the conv layers with the detection network.
Pros:

- Smart idea on sharing conv layers between RPN and R-CNN which greatly reduced the run-time of object proposals.
- State-of-the-art performance on object detection.
- Well-written paper (Really not every paper we read is catchy).

Cons:

- The training of RPN and fast R-CNN are weakly combined by alternating optimization and there is no convergence proof.
- RPN can be used independently as a region proposal method. However, it doesn't seem to outperform selective search.

(1) Dataset:

First, we need to download the required dataset as mentioned in the text which is PASCAL VOC 2007, 450M. This step can be implemented from the jupyter Notebook directly using *!wget+URL* command. After downloading the dataset, we should extract the dataset from the TAR file using the command !*tar xf+path_to_file*

The training data provided consists of a set of images; each image has an annotation file giving a bounding box and object class label for each object in one of the twenty classes present in the image.

(2) BaseLine code:

In this project I preferred to use the second proposed baseline code which is relatively hard as you mentioned in the project announcement. I downloaded the baseline code and installed the required dependencies. I also downloaded the pretrained models to test the data set on them.

I chose to work with Faster-RCNN ResNet50 FBN model (a classic and widely used two stage object detector which can be trained end-to-end, proposed in 2015.)

Model Representation:

Backbone: it is the part that transforms an image to feature maps, (in my case a ResNet-50) without the last fully connected layer.

Neck: it is the part that connects the backbone and heads. It performs some refinements or reconfigurations on the raw feature maps produced by the backbone. (As an example, in my case is Feature Pyramid Network (FPN)).

DenseHead:

It is the part that operates on dense locations of feature maps, including AnchorHead and AnchorFreeHead.

RoIExtractor:

It is the part that extracts RoIwise features from a single or multiple feature maps with RoIPooling-like operators. An example that extracts RoI features from the corresponding level of feature pyramids is SingleRoIExtractor in my case.

RoIHead:

It is the part that takes RoI features as input and make RoI-wise task-specific predictions, such as bounding box classification/regression, mask prediction.

(3) Running the code with training and validation on the server and testing the trained model on the test sets:

The default value of the number of epochs in the config files is 4 but due to the big period of time required to complete the training I had two options:

- the first is to train the model for only one complete epoch. (requires around 2/3 hours of training time)

- to load a pretrained model and complete the training process on it (requires short period of time depending on the loaded model and leads to the same results mentioned in the technical report and paper)

Therefore, I stuck to the 1st option to evaluate and test by my self and here are my results:

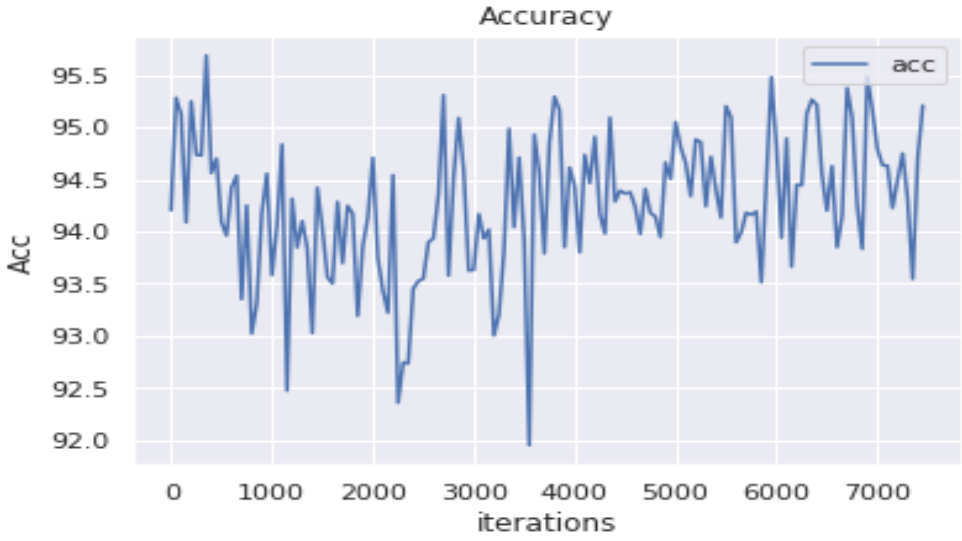
Pasting Log from console: (last few lines)

2019-11-22 20:15:59,656 - INFO - Epoch [1][7350/7517] lr: 0.01000, eta: 5:57:05, time: 0.937, data_time: 0.005, memory: 2413, loss_rpn_cls: 0.0171, loss_rpn_bbox: 0.0123, loss_cls: 0.1709, acc: 94.3105, loss_bbox: 0.1155, loss: 0.3159
2019-11-22 20:16:47,364 - INFO - Epoch [1][7400/7517] lr: 0.01000, eta: 5:56:19, time: 0.954, data_time: 0.005, memory: 2413, loss_rpn_cls: 0.0184, loss_rpn_bbox: 0.0140, loss_cls: 0.1967, acc: 93.5469, loss_bbox: 0.1289, loss: 0.3580

2019-11-22 20:17:34,859 - INFO - Epoch [1][7450/7517] lr: 0.01000, eta: 5:55:33, time: 0.950, data_time: 0.006, memory: 2413, loss_rpn_cls: 0.0174, loss_rpn_bbox: 0.0118, loss_cls: 0.1640, acc: 94.7090, loss_bbox: 0.0942, loss: 0.2875
2019-11-22 20:18:22,084 - INFO - Epoch [1][7500/7517] lr: 0.01000, eta: 5:54:46, time: 0.944, data_time: 0.005, memory: 2413, loss_rpn_cls: 0.0138, loss_rpn_bbox: 0.0086, loss_cls: 0.1584, acc: 95.2070, loss_bbox: 0.0863, loss: 0.2670


Loss
loss_rpn_cls
loss_rpn_bbox
Loss_RPN
0.12
0.10
0.08
0.06
0.04
0.02
0
1000
2000
3000
4000
5000
6000
7000
iterations


Loss
loss_cls
loss_bbox
Loss cls & bbox
0.5
0.4
0.3
0.2
0.1
0
1000
2000
3000
4000
5000
6000
7000
iterations


Accuracy
acc
Acc
95.5
95.0
94.5
94.0
93.5
93.0
92.5
92.0
0
1000
2000
3000
4000
5000
6000
7000
iterations

Comments on the results:

By comparing the obtained results above with results stated in the paper and the technical report we can notice the results are almost similar to each other's if we compare only by the number of iterations (in my case, only one epoch or 7500 iterations) and this is logical because I used the same model and hyper parameters but I stopped the training process after completing the first epoch out of four done by them.

Optional Task 1:

Adding the path to the new dataset:

```
data = dict(
    imgs_per_gpu=2,
    workers_per_gpu=2,
    train=dict(
        type='RepeatDataset',
        times=3,
        dataset=dict(
            type=dataset_type,
            ann_file=[
                data_root + 'VOC2007/ImageSets/Main/trainval.txt',
                data_root + 'VOC2012/ImageSets/Main/trainval.txt'
            ],
            img_prefix=[data_root + 'VOC2007/', data_root + 'VOC2012/'],
            pipeline=train_pipeline)),
    val=dict(
        type=dataset_type,
        ann_file=data_root + 'VOC2007/ImageSets/Main/test.txt',
        img_prefix=data_root + 'VOC2007/',
        pipeline=test_pipeline),
    test=dict(
        type=dataset_type,
```

```
        ann_file=data_root + 'VOC2007/ImageSets/Main/test.txt',
        img_prefix=data_root + 'VOC2007/',
        pipeline=test_pipeline))
```

Main Results

	training data	*Train time (s/iter)*	*test data*	*mAP*	*time/img*
Faster RCNN, resnet 50	VOC 2007 trainval	0.333	VOC 2007 test	69.9%	0.950
Faster RCNN, resnet 50	VOC 2007 trainval + 2012 trainval	0.353	VOC 2007 test	73.2%	0.943
Faster RCNN, resnet 50	VOC 2012 trainval	0.333	VOC 2007 test	67.0%	0.950

Note: The mAP results are subject to hyperparameters and changes with the number of epochs and the learning rate

As we can see from the results and the accuracy of our trained model improved by 3% after train on the new dataset VOC2012 in addition to the old one. By comparing my results with the other results from papers (73.8%), we can say it's almost the same.

README DOC follows……

Coding Project 2 - Youshaa Murhij

Required Task

First, we specify the directory where to doanload the repository

```
%cd /content/drive/My\ Drive/
```

Cloning the repository from Github, which contians teh baseline code for our task

```
!git clone https://github.com/open-mmlab/mmdetection.git
```

```
Cloning into 'mmdetection'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 5492 (delta 1), reused 1 (delta 0), pack-reused 5483
Receiving objects: 100% (5492/5492), 5.07 MiB | 8.20 MiB/s, done.
Resolving deltas: 100% (3710/3710), done.
Checking out files: 100% (384/384), done.
```

Going to the path of the repository

```
%cd /content/drive/My\ Drive/mmdetection
```

Installing the required dependencies and libraries after updating the the versions of them because

```
!pip install -r requirements.txt
```

Setting up the project by runing the python code setup.py

```
!python setup.py develop
```

Creating an new directory to store the dowloaded dataset

```
!mkdir data
%cd /content/drive/My\ Drive/mmdetection/data
```

```
%cd ..
!ls
```

Downloading the dataset Pascal VOC from the offical site as compressed TAR files

```
!wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.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-2007.tar
```

Extracting the real dataset from the compressed files and storing them into a VOCdevkit folder as

```
!tar  xf VOCtrainval_06-Nov-2007.tar
!tar  xf VOCtest_06-Nov-2007.tar
!tar  xf VOCdevkit_08-Jun-2007.tar
```

This is an example how we can use the testing python file to test a trained model. it needs a confi output file for results and evaluation metrics and we can choose whether to show the results or nc

```
!python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out ${RESULT_FILE}] [--eval
```

Here, I created a checkpoints folder to save the pretrained models and checkpoints and downlode resnet 50 FBN as recommended in the task

```
%cd ..
!mkdir checkpoints
%cd checkpoints
!ls
!wget https://s3.ap-northeast-2.amazonaws.com/open-mmlab/mmdetection/models/faster_rcnn
```

```
%cd /content/drive/My\ Drive/mmdetection
```

Here, I tested the pretrained model on the testing data and stored all the results in pkl output file tc

```
!python tools/test.py configs/pascal_voc/faster_rcnn_r50_fpn_1x_voc0712.py \
    checkpoints/faster_rcnn_r50_fpn_1x_20181010-3d1b3351.pth \
    --out Results.pkl
```

```
terminal width is too small (0), please consider widen the terminal for better progre
[>>>>>>>>>>] 4952/4952, 4.0 task/s, elapsed: 1226s, ETA:     0s
writing results to Results.pkl
```

```
%matplotlib notebook
```

Here, I choosed to show the output without storing the results

```
!python tools/test.py configs/pascal_voc/faster_rcnn_r50_fpn_1x_voc0712.py \
    checkpoints/faster_rcnn_r50_fpn_1x_20181010-3d1b3351.pth \
    --show
```

Downloading some other pretrainded models from Model ZOO as mentioned in the repository and

```
https://s3.ap-northeast-2.amazonaws.com/open-mmlab/mmdetection/models/faster_rcnn_r50_f
```

```
!wget https://s3.ap-northeast-2.amazonaws.com/open-mmlab/mmdetection/models/ssd300_coco
```

```
!wget https://s3.ap-northeast-2.amazonaws.com/open-mmlab/mmdetection/models/ssd512_coco
```

RPN : R-50-FPN pytorch 1x

```
!wget https://s3.ap-northeast-2.amazonaws.com/open-mmlab/mmdetection/models/rpn_r50_fpn
```

Here, I invoked the training function on the smae model : FASTER RCNN RESNET 50 FBN and store to use it later to plot the training results figures as required

```
!python tools/train.py  configs/pascal_voc/faster_rcnn_r50_fpn_1x_voc0712.py >out.txt
#--resume_from checkpoints/faster_rcnn_r50_fpn_1x_20181010-3d1b3351.pth
```

```
!pip install seaborn
```

Here, we can plot loss/mAP curves given a training log file

```
!python tools/analyze_logs.py plot_curve log.json --keys loss_cls --legend loss_cls
```

```
!python tools/voc_eval.py  Results.pkl configs/pascal_voc/faster_rcnn_r50_fpn_1x_voc071
```

```
!python demo/webcam_demo.py configs/pascal_voc/faster_rcnn_r50_fpn_1x_voc0712.py \
    checkpoints/faster_rcnn_r50_fpn_1x_20181010-3d1b3351.pth
```

Here, we read the sored reults file from the training process and define the paprameters that we w of iterations as in the baseline code

```
infile='/content/drive/My Drive/mmdetection/out.txt'
loss_rpn_cls=[]
loss_rpn_bbox=[]
loss_cls=[]
acc=[]
loss_bbox=[]
loss=[]
with open(infile) as f:
  for l in f:
    loss_rpn_cls.append(float(l[140:146]))
    loss_rpn_bbox.append(float(l[163:169]))
    loss_cls.append(float(l[181:187]))
    acc.append(float(l[194:201]))
    loss_bbox.append(float(l[214:220]))
    loss.append(float(l[228:235]))
```

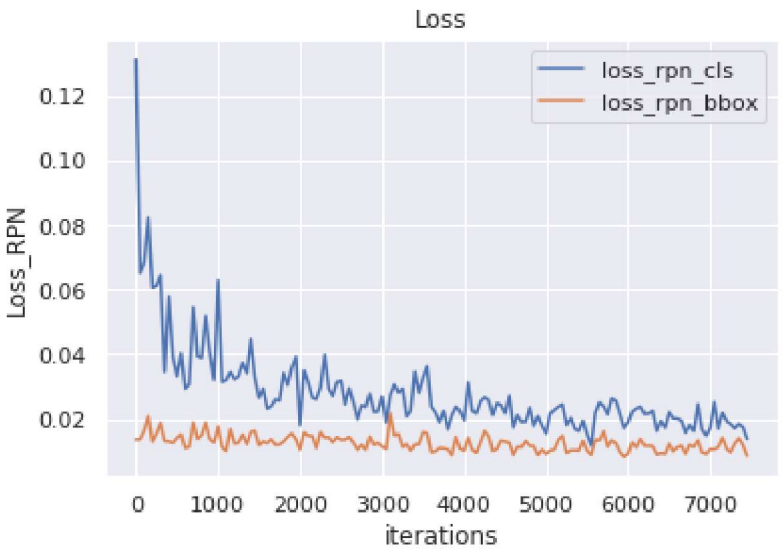
```
%matplotlib inline
```

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

```
x= range(0,7500,50)
```

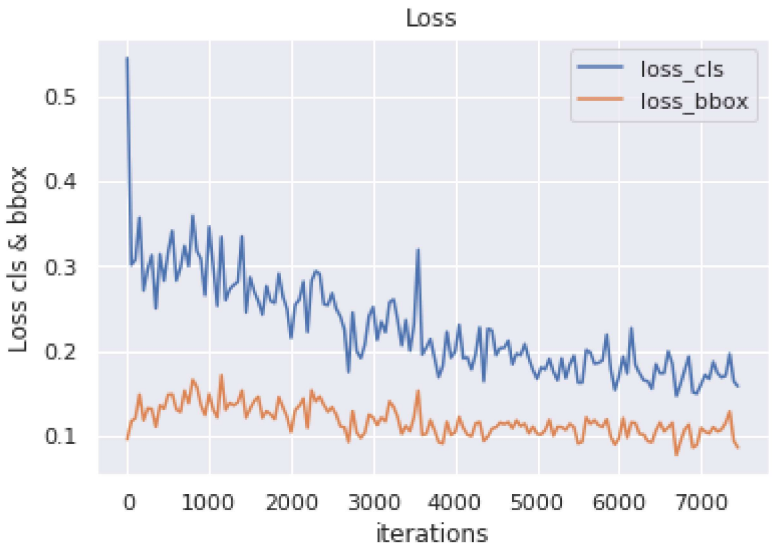
Plotting the Loss-RPN

```
plt.xlabel('iterations')
plt.ylabel('Loss_RPN')
plt.title('Loss')
plt.plot(x,loss_rpn_cls)
plt.plot(x,loss_rpn_bbox)
plt.legend(['loss_rpn_cls','loss_rpn_bbox'],loc='upper right');
plt.show()
```



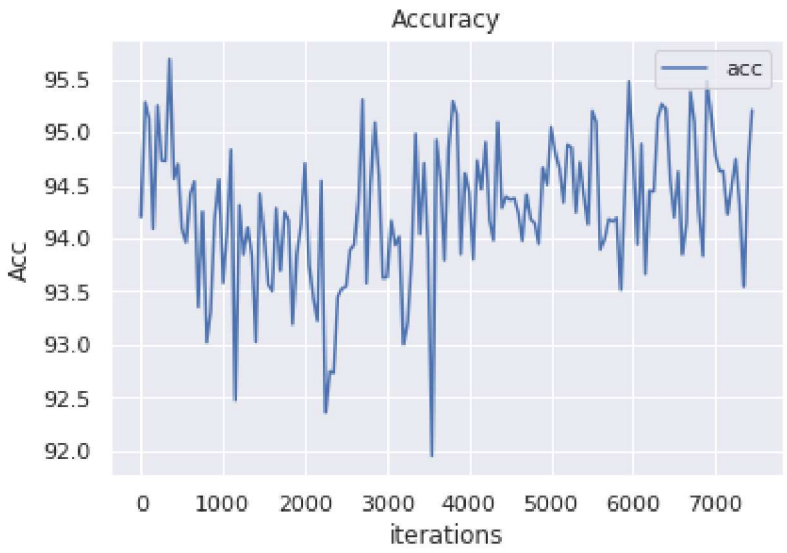
Plotting Loss cls & bbox

```
plt.xlabel('iterations')
plt.ylabel('Loss cls & bbox')
plt.title('Loss')
plt.plot(x,loss_cls)
plt.plot(x,loss_bbox)
plt.legend(['loss_cls','loss_bbox'],loc='upper right');
plt.show()
```



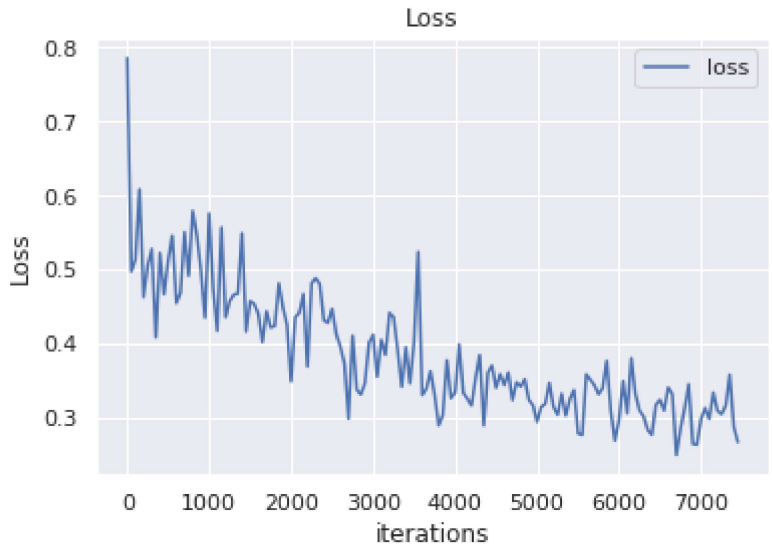
Plotting the accuracy

```
plt.xlabel('iterations')
plt.ylabel('Acc')
plt.title('Accuracy')
plt.plot(x,acc)
#plt.plot(loss)
plt.legend(['acc'],loc='upper right');
plt.show()
```



Plotting the Loss

```
plt.xlabel('iterations')
plt.ylabel('Loss')
plt.title('Loss')
plt.plot(x,loss)
plt.legend(['loss'],loc='upper right');
plt.show()
```




```
!python tools/voc_eval.py results.pkl configs/dcn/faster_rcnn_mdpool_r50_fpn_1x_VOC.py
```

Optional Task 1

Directing to the dataset path to add the new dataset

```
%cd /content/drive/My\ Drive/mmdetection/data
```

Dowloading the Pascal VOC2012 dataset from the official site

```
!wget -c http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar
```

Extracting the dataset from the compressec file

```
!tar xf VOCtrainval_11-May-2012.tar
```

```
!rm VOCtrainval_11-May-2012.tar
```

Here, I retarined the previous model using the new downloaded data set VOC2012 but I prefred to log data to txt file to discuss and plot the reuired figures

```
!python tools/train.py  configs/pascal_voc/faster_rcnn_r50_fpn_1x_voc0712.py >out2.txt\
                --resume_from checkpoints/faster_rcnn_r50_fpn_1x_20181010-3d1b3351.pth
```

Testing on the testset after training and saving the results in pkl output file

```
!python tools/test.py configs/pascal_voc/faster_rcnn_r50_fpn_1x_voc0712.py \
    checkpoints/faster_rcnn_r50_fpn_1x_20181010-3d1b3351.pth \
    --out Results.pkl
```

```
        out results.pkl
```