CSI4106 Introduction to Artificial Intelligence

Project Final Report

Group #21

Project Title:

Object Detection using Faster R-CNN

Project type:

Type 3:

In-depth understanding of a solution approach to an AI problem, with implementation and tests

Group members:

Wentao Yang	8957505	wyang010@uottawa.ca
Yi Zhao	8650881	yzhao156@uottawa.ca
Junbo Tang	8639271	btang076@uottawa.ca

Project GitHub link:

https://github.com/yzhao156/CSI4106_Faster_RCNN

1. Timesheet

Team member	Task	Hours spent	Challenge
Wentao Yang	Read the paper (Ren, He, Girshick, & Sun, 2017)	3	New ideas not seen in class, many new structures and methods, long and hard to understand
	Find the most appropriate code about Faster R-CNN from GitHub (see Faster R-CNN link in the Reference), and make it works on our machines, then write a document and teach other members how to use it	2	
	Find the way to set up our Google Cloud Platform (GCP) to provide computing power for our group, link it to SSH tools and PyCharm, then write a document and teach other members how to use it	7	Faced many problems when setting up the GCP and using different software to connect the GCP (especially there are some differences between the same things on Windows and Mac)
	Find the most appropriate code about Fast R-CNN (our baseline approach) from GitHub (see Faster R-CNN link in the Reference), make it works and modify it to make it can run on the same dataset as our Faster R-CNN, and evaluate its result	Over 15 hours	The code is old and written based on Caffe, which is another deep learning framework that I am not familiar. Caffe is much harder to install and the version of Caffe used in this code has many problems that is very hard to solve
	Learn to have a deeper understanding of Convolutional Neural Networks (CNN), because to understand the principle of Faster R-CNN, a decent understanding of CNN is required	2	Our course only covered an introduction of CNN, need to learn more
	Learn how to use TensorFlow	4	I have used it over two years ago, not very familiar with the new version and have to revise how to use it
	Understand the Faster R-CNN code and write comments of some parts of it	4	The algorithm and the code are much more complex than any other machine learning algorithm and code I have seen before
	Modified the codes for demo to meet our requirements	3	Need to understand the logic between different files, and there are some problems in the paths written in the code
	Implemented the codes for	5	I implemented this functionality by my self,

	calculating the metrics for our code (including calculationg mAP, etc.)		there is no such solution for this implementation on the Internet, I solved it by studying the logic between all of the related files, referencing another Caffe-version implementation of Faster R-CNN. Many people on the Internet think it is impossible to achieve this functionality using that version of implementation
	Try different batch sizes and train the models corresponding to the different batch sizes, then replace all of the files required and run demo.py for each of them	2	Not difficult, but requires times to do them
	Try to modify the optimizer	1.5	Faced some weird problems, but decided stop spending more time on it
	Answering questions and helping solving problems for other members	5	Other two group members have less experience in analyzing codes and understanding in neural networks, often need to help them from stuck by some issues, and give some ideas for next steps
	Writing report	Over 15 hours	Need to record many stuffs, also need to verify and edit other members' part to roughly ensure the correctness of theory and gramma. Did not sleep for 2 separate nights.
Yi Zhao	Read paper (Ren, He, Girshick, & Sun, 2017), pay more attention from the ROI pooling to the output of the Faster R-CNN	5	new ideas not seen in class, many new structures and methods, long and hard to understand
	Config the GCP's VM with the help of Wentao Yang, learn the basics commands of Linux and transfer data through git.	2	No UI in GCP's VM, can only use git to modify the file in the VM, more complex to modify and learned how to push large files like checkpoints on git.
	Learn the principle of CNN and how to use TensorFlow to understand the implantation code of Faster R-CNN	5	Only doing research when there is a part I don't understand in Faster R-CNN makes my understanding in Faster R-CNN very inconsistent. Therefore, I have to learn TensorFlow systematically.
	Understand each line of code of Faster R-CNN's RPN network and write comment on the code with a short summary.	7	Even through there are some explanations on the internet, there are no explanations on each line of code, to understand the code I have to do a lot of research while writing my comments.
	Modify the learning rate to half and modify the max iterations and run it	1	

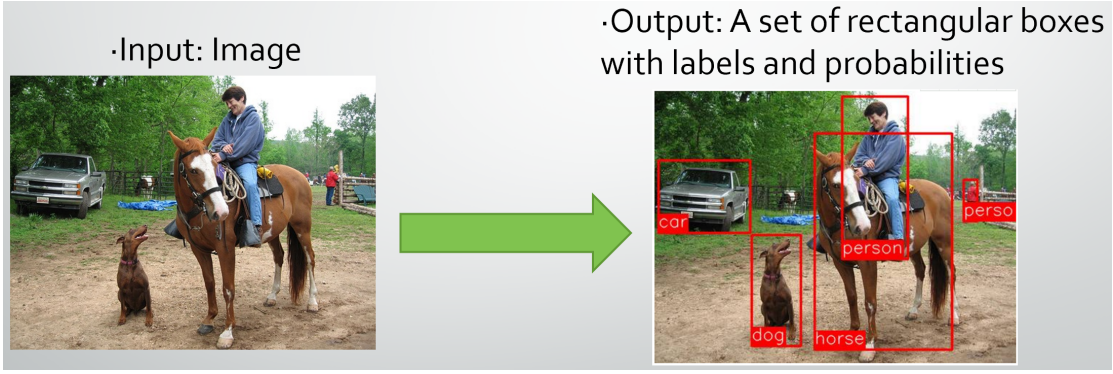
	on GCP.		
	Try to modify the net from vgg16 to vgg19 and resnet152 reference Pre-trained Models on GitHub.	6	When I do resnet152, I find a value error that I haven't solve, I did research and believe it's about the version is different than what I modify, but not knowing which package and which version I should try.
	Write README.md, Learn details from my partners and do knowledge transforms.	3	Have to know the details for all parts in Faster R-CNN rather than just my part(RPN layers.)
	Report writing	3	
Junbo Tang	Read paper (Ren, He, Girshick, & Sun, 2017), focus more on the Region Proposal Network (RPN) part	5	new ideas not seen in class, many new structures and methods, long and hard to understand
	Understand the Faster R-CNN code and write comments for some parts of it	5	
	Implementing the evaluation method (mAP) for Faster R-CNN and baseline and editing the evaluation code	5	Set the environment; let the evaluation code fit with the Faster R-CNN code; search and study the principle of calculating PR line, APs and mAP
	Learn more about CNN and how to use TensorFlow	6	This is the first time I use TensorFlow, I do not know about the functions and parameters.
	Writing the summary for the paper and share my understandings to other members	4	Describe the complex idea in my own words
	Setup Google Cloud Platform (GCP) under Wentao's guidance	1	The MacOS is different from Windows
	Writing report	4	

2. Introduction

Problem studied

We studied Faster R-CNN, which one of the most powerful object detection algorithms today.

Object detection deals with detecting instances of semantic objects of a certain class in digital images and videos, in our project, we focus on the work on images. Basically, given an image, our goal is to use rectangular boxes to circle all of the objects in the image, and for every box give the label of the object it circles with the probability the object belongs to the label.



Context

With the development of computer technologies and computing hardwares, nowadays object detection is widely used in computer vision tasks such as face detection, face recognition, self-driving cars and video object co-segmentation. Thus, today object detection is a very promising and hot research area.

We focus on Faster R-CNN meanly because it is a very classic object detection algorithm and many of its structures have been referenced by the algorithms invented in recent years. We would like to let this project to be our starting point in the area of object detection, to force us to understand the process of object detection. At the beginning we want to let each member work on a different approach (Faster R-CNN, SSD and YOLO, they are all the most powerful object detection algorithms today) and then compare the result, so we were doing type 2. But just before the day of the deadline of submitting the proposal, we found the workload may be too heavy to afford for other members, and considered we need to modify the code. Thus, we decided to do type 3 and focus on one classic algorithm, which is Faster R-CNN. We thought the workload should be lighter and we would have a chance to have a better understanding of Faster R-CNN.

Goals:

(1) To have an in-depth understanding of Faster R-CNN object detection algorithm.

(2) To learn how to use TensorFlow machine learning platform, and to have an experience of implementing Faster R-CNN for object detection tasks.

(3) To learn how to use object detection datasets and how to evaluate the performance of trained models for object detection tasks.
(4) To make some modifications to the original Faster R-CNN algorithm, in the hope of improving the performance.

We are trying to achieve:
(1) Implementing Faster R-CNN algorithm using TensorFlow for object detection tasks.
(2) Trying to make some modifications and test the performance, in the hope of improving the performance of the original algorithm.
(3) Having an in-depth understanding of Faster R-CNN object detection algorithm, and being able to solve machine learning tasks using TensorFlow.

Link to AI

Object detection is a research hotspot of AI, and Faster R-CNN is a very classic object detection algorithm. In this course, the topics about the metrics to evaluate a model (including precision, recall, accuracy, etc.), Neural Networks and Deep learning architectures (Application to vision) will give some helps for us.

Why it is important

Object detection is currently a very hot area of computer vision research, a powerful and efficient object detection algorithm can provide tremendous helps for human's work. Also, in recent years many powerful object detection algorithms were invented, including SSD (Single Shot MultiBox Detector), R-FCN (Region-based Fully Convolutional Networks), YOLO (You Only Look Once) and many improved versions of them. Faster R-CNN is a very classic object detection algorithm, and many of its structures have been referenced by the algorithms invented in recent years. Therefore, for the purpose to get started in the area of object detection and to learn how to use TensorFlow machine learning platform, we think it is a good choice for us to start with a deep understanding of Faster R-CNN.

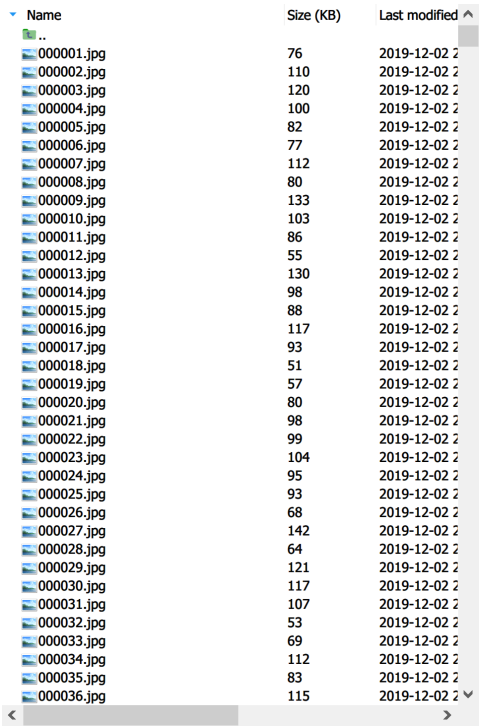
3. Dataset description / Features

We are using a very classical dataset: PASCAL VOC 2007, it's a competition and VOC 2007 contains 9963 annotated pictures with 20 classes of objects.

Link:
http://host.robots.ox.ac.uk/pascal/VOC/voc2007/

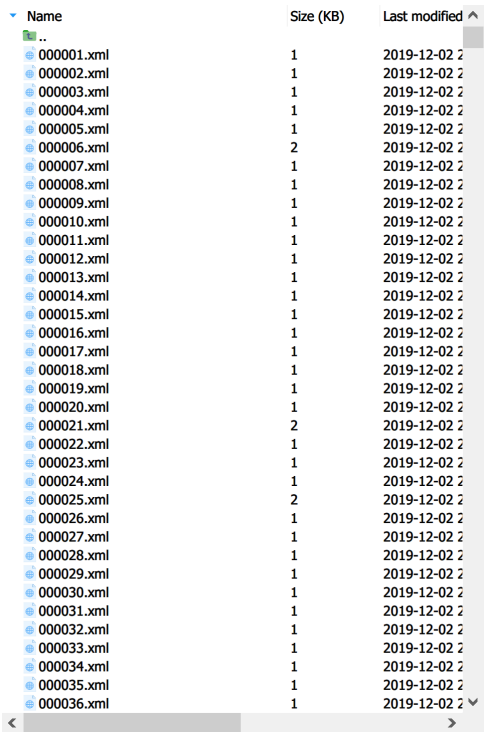
The dataset is separated to 5011 training images and 4952 test images.

Folder VOCDevkit2007\VOC2007\JPEGImages stores all of the image files:



Name	Size (KB)	Last modified
..		
000001.jpg	76	2019-12-02 2
000002.jpg	110	2019-12-02 2
000003.jpg	120	2019-12-02 2
000004.jpg	100	2019-12-02 2
000005.jpg	82	2019-12-02 2
000006.jpg	77	2019-12-02 2
000007.jpg	112	2019-12-02 2
000008.jpg	80	2019-12-02 2
000009.jpg	133	2019-12-02 2
000010.jpg	103	2019-12-02 2
000011.jpg	86	2019-12-02 2
000012.jpg	55	2019-12-02 2
000013.jpg	130	2019-12-02 2
000014.jpg	98	2019-12-02 2
000015.jpg	88	2019-12-02 2
000016.jpg	117	2019-12-02 2
000017.jpg	93	2019-12-02 2
000018.jpg	51	2019-12-02 2
000019.jpg	57	2019-12-02 2
000020.jpg	80	2019-12-02 2
000021.jpg	98	2019-12-02 2
000022.jpg	99	2019-12-02 2
000023.jpg	104	2019-12-02 2
000024.jpg	95	2019-12-02 2
000025.jpg	93	2019-12-02 2
000026.jpg	68	2019-12-02 2
000027.jpg	142	2019-12-02 2
000028.jpg	64	2019-12-02 2
000029.jpg	121	2019-12-02 2
000030.jpg	117	2019-12-02 2
000031.jpg	107	2019-12-02 2
000032.jpg	53	2019-12-02 2
000033.jpg	69	2019-12-02 2
000034.jpg	112	2019-12-02 2
000035.jpg	83	2019-12-02 2
000036.jpg	115	2019-12-02 2

Folder VOCDevkit2007\VOC2007\Annotations stores all of the annotations of each picture, every image in VOCDevkit2007\VOC2007\JPEGImages has an annotation, stored as a .xml file, as the following shows:



Name	Size (KB)	Last modified
..		
000001.xml	1	2019-12-02 2
000002.xml	1	2019-12-02 2
000003.xml	1	2019-12-02 2
000004.xml	1	2019-12-02 2
000005.xml	1	2019-12-02 2
000006.xml	2	2019-12-02 2
000007.xml	1	2019-12-02 2
000008.xml	1	2019-12-02 2
000009.xml	1	2019-12-02 2
000010.xml	1	2019-12-02 2
000011.xml	1	2019-12-02 2
000012.xml	1	2019-12-02 2
000013.xml	1	2019-12-02 2
000014.xml	1	2019-12-02 2
000015.xml	1	2019-12-02 2
000016.xml	1	2019-12-02 2
000017.xml	1	2019-12-02 2
000018.xml	1	2019-12-02 2
000019.xml	1	2019-12-02 2
000020.xml	1	2019-12-02 2
000021.xml	2	2019-12-02 2
000022.xml	1	2019-12-02 2
000023.xml	1	2019-12-02 2
000024.xml	1	2019-12-02 2
000025.xml	2	2019-12-02 2
000026.xml	1	2019-12-02 2
000027.xml	1	2019-12-02 2
000028.xml	1	2019-12-02 2
000029.xml	1	2019-12-02 2
000030.xml	1	2019-12-02 2
000031.xml	1	2019-12-02 2
000032.xml	1	2019-12-02 2
000033.xml	1	2019-12-02 2
000034.xml	1	2019-12-02 2
000035.xml	1	2019-12-02 2
000036.xml	1	2019-12-02 2

Every picture has an annotation, as an example, the annotation for 000005.jpg is as the following table shows, each row contains the names of the database and dataset it belongs to,

and the information of its source and its pose. The most important information is the position and the size of a box, and the number of rows indicates the number of boxes of a picture.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	folder	filename	database	annotation	image	flickrid	flickrid2	name	width	height	depth	segmented	name3	pose	truncated	difficult	xmin	ymin	xmax	ymax
2	VOC2007	000005.jpg	The VOC2007 Database	PASCAL VOC2007	flickr	325991873	archintent louisville	?	500	375	3	0	chair	Rear	0	0	263	211	324	339
3	VOC2007	000005.jpg	The VOC2007 Database	PASCAL VOC2007	flickr	325991873	archintent louisville	?	500	375	3	0	chair	Unspecified	0	0	165	264	253	372
4	VOC2007	000005.jpg	The VOC2007 Database	PASCAL VOC2007	flickr	325991873	archintent louisville	?	500	375	3	0	chair	Unspecified	1	1	5	244	67	374
5	VOC2007	000005.jpg	The VOC2007 Database	PASCAL VOC2007	flickr	325991873	archintent louisville	?	500	375	3	0	chair	Unspecified	0	0	241	194	295	299
6	VOC2007	000005.jpg	The VOC2007 Database	PASCAL VOC2007	flickr	325991873	archintent louisville	?	500	375	3	0	chair	Unspecified	1	1	277	186	312	220

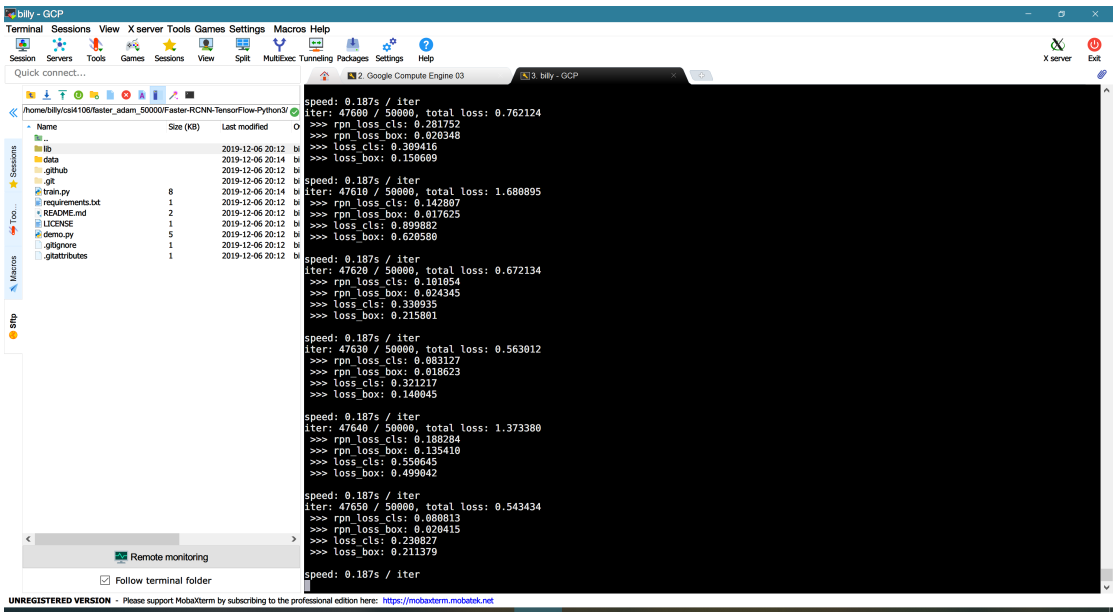
PASCAL VOC Competition is a completion including several tasks, but we only focuses on the object detection task, thus the data we needed are only the images and its annotations introduced above.

4. Setup the Google Cloud Platform

To obtain enough computing power to train our model, and to have a choice to familiar with using remote Linux platform, we decided to use Google Cloud Platform (GCP).

To use the GCP, Wentao Yang first spent a whole afternoon and evening to find out how to make every thing works on the GCP, including how to link to the GCP through local SSH software (including MobaXterm (on Windows only), Termius (on Macs only) and Pycharm, etc.) using RSA public key and private key. The he recorded what he did and taught other members how to set up their GCP and connect to it successfully. So finally our group has three platforms to use.

The following picture is a screenshot of training our model using the GCP, linked to the GCP through MobaXterm in Windows.



Our platform configurations:

Hardware:

OS: Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1050-gcp x86_64)

4 vCPUs

15GB RAM

GPU: NVIDIA Tesla V100 with 16GB VRAM

Software:

Python 3.7 with Anaconda

TensorFlow-GPU, version 1.15

openCV 4.1.2 + its contrib modules

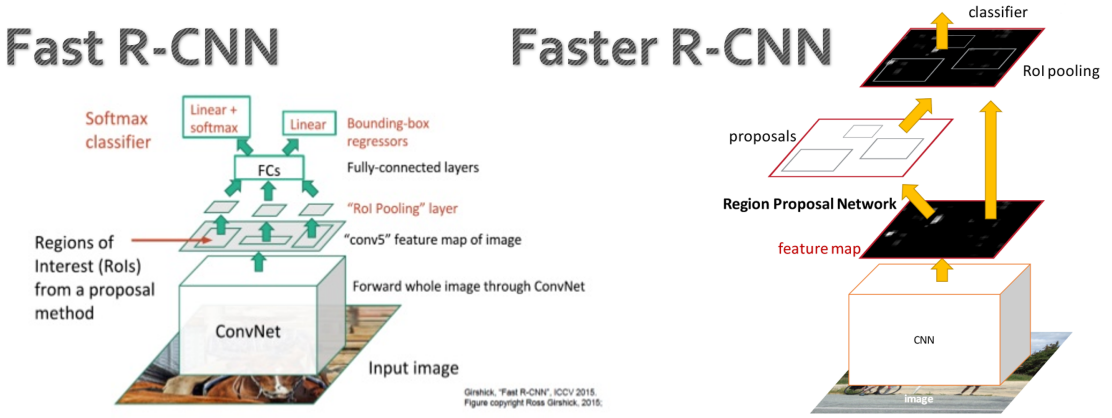
CUDA 10.0

cuDNN 7.6.5

5. Baseline Approach: Fast R-CNN

Because Faster R-CNN is invented based on Fast R-CNN, so we pick Fast R-CNN as our baseline.

The code we chose to use is from https://github.com/rbgirshick/fast-rcnn.git, which is a very famous implementation of Fast R-CNN. This implementation is written in Python and based on Caffe frameware, but the Caffe version used in this code is very old and because this repository has not been maintained for several years, we did meet a huge number of difficulties until successfully made some progresses.



Comparing with Faster R-CNN, because Fast R-CNN uses Selective Search to generate region proposals, the superiority of GPU cannot be utilized. The time for the whole detection process is around two seconds but generating region proposals already takes around two seconds, so the process of generating region proposals is the main bottleneck. The greatest improvement in Faster R-CNN is generating Region Proposals using RPN (Region Proposal Network), a network, instead of using Selective search. And thus the CNN for detection can also be shared with the use of generating Region Proposals.

After that, Region Proposals can be generated much more efficiently, the number of Region Proposals is reduced from two thousand to three hundred, and the quality of each Region Proposal can also be greatly improved.

Because our project mainly focuses on Faster R-CNN and Faster R-CNN is already very complex to be understanded and implemented, with the approval from the professor, we are allowed to just find an implementation of the baseline approach and make it works to be run.

5.1 How to successfully install and run demo.py in /tools folder

We have tried the Google Cloud Platform first, whose base environment is Ubuntu 16.04 and using Python 3.7 in Anaconda, with CUDA 10.0 and cuDNN 7.6.5. Maybe because the code uses the first generation of Caffe that prefer python 2.7, and our CUDA and cuDNN version is too high for the Caffe frameware, I have spent over six hours on fixing the problem to try to run the code on my Google Cloud Platform, but there are still some wired errors that were very difficult to fix.

Thus, I decided to use my old gaming laptop, with Ubuntu 14.04 operating system and python 2.7, and with CUDA 8.0 and cuDNN 5.0.5. The GPU for this laptop is NVIDIA GTX 870M, with 6GB VRAM.

Generally, as long as you are running a Ubuntu operating system with Python version 2.7 and you have a GPU with enough VRAM, you should run the code successfully after following steps.

Packages maybe required:
sudo apt-get install libprotobuf-dev
sudo apt-get install libleveldb-dev
sudo apt-get install libsnappy-dev
sudo apt-get install libopencv-dev
sudo apt-get install libhdf5-serial-dev
sudo apt-get install protobuf-compiler
sudo apt-get install -no-install-recommends libboost-all-dev
sudo apt-get install libatlas-base-dev
sudo apt-get install python-dev
sudo apt-get install libgflags-dev
sudo apt-get install libgoogle-glog-dev
sudo apt-get install liblmdb-dev
sudo apt-get install python-pip

(1) Open a terminal, find a good place and create a folder to store the project for Fast R-CNN.

(2) Download the source code: git clone --recursive https://github.com/rbgirshick/fast-rcnn.git

(3) Download the pretrained model using a given script:
cd fast-rcnn
./data/scripts/fetch_fast_rcnn_models.sh

Then the model will be stored in /data/fast_rcnn_models

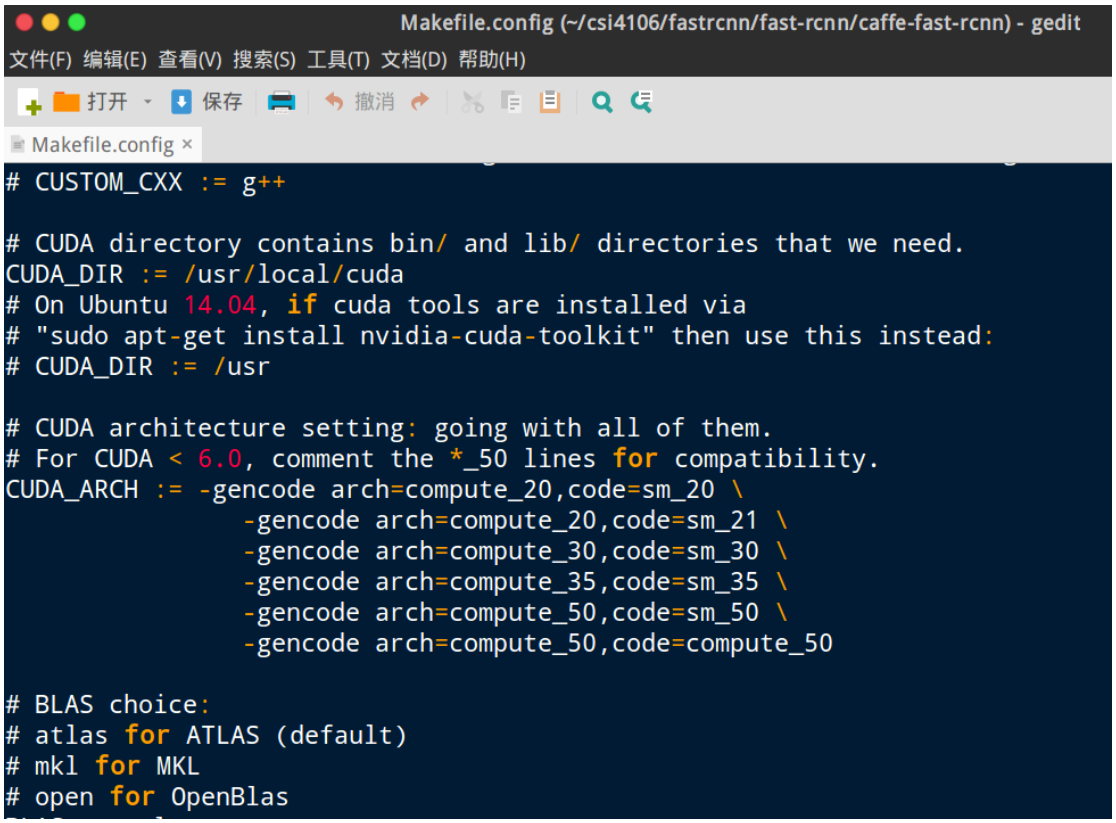
(4) Now, replace the caffe-fast-rcnn folder with the caffe-fast-rcnn from https://github.com/rbgirshick/py-faster-rcnn, to get the complete repository, use command git -r https://github.com/rbgirshick/py-faster-rcnn in a directory that you want to store the repository.

(5) cd caffe-fast-rcnn
cp Makefile.config.example Makefile.config
sudo gedit Makefile.config

In the editor, uncomment the following lines:
USE_CUDNN = 1
WITH_PYTHON_LAYER = 1
USE_PKG_CONFIG = 1

Add path “/usr/include/hdf5/serial” (no quotation marks) to INCLUDE_DIRS
Add path “/usr/lib/x86_x64-linux-gnu/hdf5/serial” to LIBRARY_DIRS

Then delete the first two lines of CUDA_ARCH, the result will be like:



(6) Now, compile Cython module:
cd fast-rcnn/lib
make

(7) Open the folder: cd caffe-fast-rcnn
After that, replace all of the following files in the Fast R-CNN repository by the corresponding files from the repository of the latest version of Caffe (from https://github.com/BVLC/caffe.git):

/include/caffe/util/cudnn.hpp
copy the whole /include/caffe/layers folder from the latest version of Caffe
all of the files name start with cudnn in /src/caffe/layer
/src/caffe/util/cudnn.cpp

And the following three files:
pooling_layer.hpp
pooling_layer.cpp
pooling_layer.cu

At the last, add the following code to the “message PoolingParameter” in caffe.proto:

```
enum RoundMode {
      CEIL = 0;
      FLOOR = 1;
  }
  optional RoundMode round_mode = 13 [default = CEIL];
```

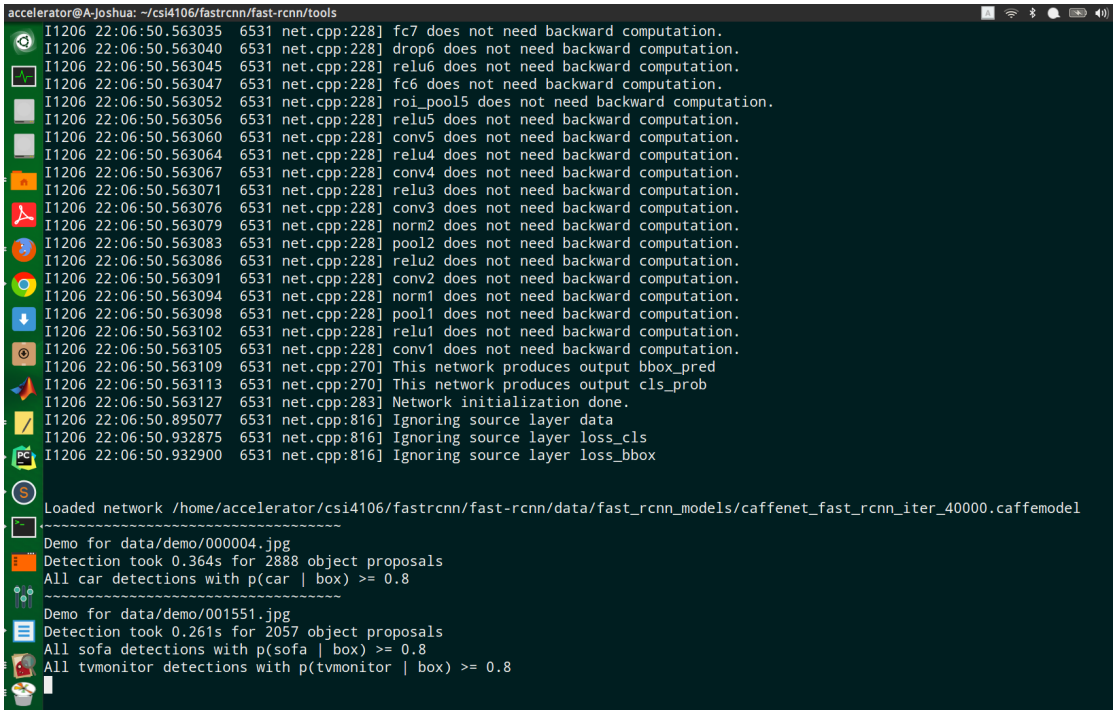
(8) make files in caffe-fast-rcnn:
make
make pycaffe

Then, the installation should be successfully completed.

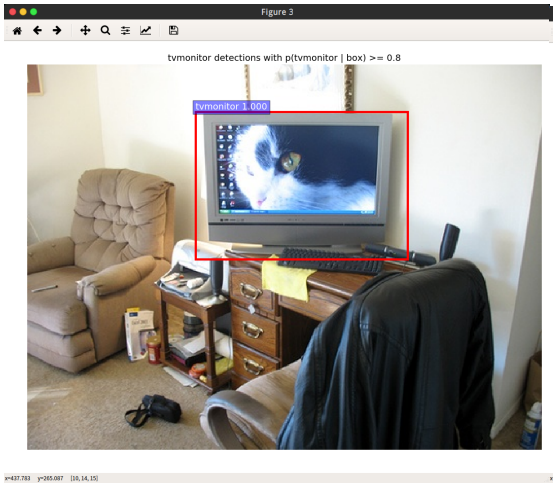
(9) To run the demo, open the folder:
cd fast-rcnn/tools

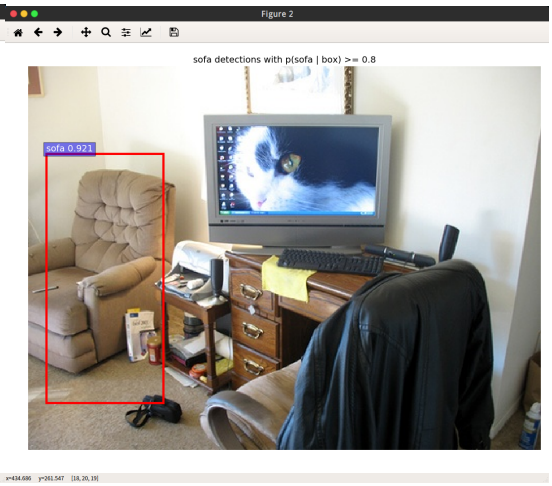
Replace the demo.py with our edited version in our GitHub repository, then run:
python demo.py --net caffenet

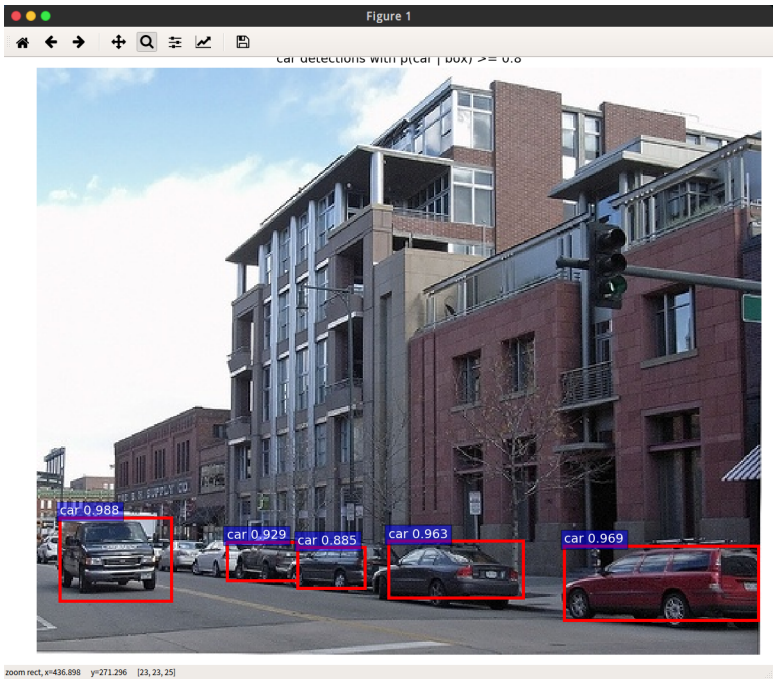
The output should be:



And the boxes are drawn on the demo images and show:







2. Efforts to calculate the mAP

We have not finished this part yet, although the logic between the files related to calculating the mAP between this code and the Faster R-CNN code we used are similar, but there are many difference between the detailed implementation in codes, we have tried modified the code many times, but we still cannot invoke the function to calculate mAPs successfully.

```
I1206 22:16:06.714882  6834 net.cpp:228] pool1 does not need backward computation.
I1206 22:16:06.714886  6834 net.cpp:228] relu1 does not need backward computation.
I1206 22:16:06.714890  6834 net.cpp:228] conv1 does not need backward computation.
I1206 22:16:06.714895  6834 net.cpp:270] This network produces output bbox_pred
I1206 22:16:06.714900  6834 net.cpp:270] This network produces output cls_prob
I1206 22:16:06.714916  6834 net.cpp:283] Network initialization done.
I1206 22:16:07.043367  6834 net.cpp:816] Ignoring source layer data
I1206 22:16:07.081125  6834 net.cpp:816] Ignoring source layer loss_cls
I1206 22:16:07.081152  6834 net.cpp:816] Ignoring source layer loss_bbox


Loaded network /home/accelerator/csi4106/fastrcnn/fast-rcnn/data/fast_rcnn_models/caffenet_fast_rcnn_iter_40000.caffemodel
Traceback (most recent call last):
  File "demo.py", line 155, in <module>
    test_net(net, imdb)
  File "/home/accelerator/csi4106/fastrcnn/fast-rcnn/tools/../lib/fast_rcnn/test.py", line 268, in test_net
    output_dir = get_output_dir(imdb, net)
  File "/home/accelerator/csi4106/fastrcnn/fast-rcnn/tools/../lib/fast_rcnn/config.py", line 143, in get_output_dir
    return osp.join(path, net.name)
AttributeError: 'Net' object has no attribute 'name'
accelerator@A-Joshua:~/csi4106/fastrcnn/fast-rcnn/tools$
```

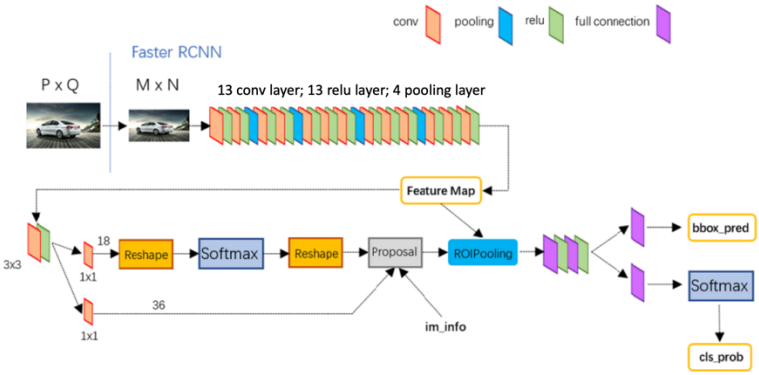
6. Advanced Approach: Faster R-CNN

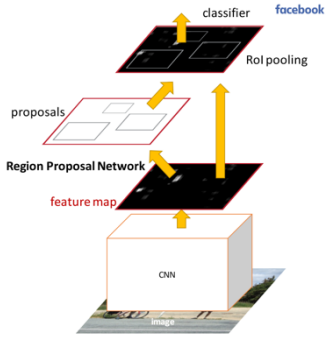
6.1 The architecture of Faster R-CNN

As project type 3, at the beginning of the project, we first read the paper (Ren, He, Girshick, & Sun, 2017) to have a good understanding of the algorithm.

Since Faster R-CNN has been introduced to public for several years, it is not only an algorithm for object detection, it is more and more to be like a frame for object detection tasks. The detection time and accuracy of faster R-CNN are faster and higher than all of the previous algorithm, because it uses Region Proposal Network (RPN) to get the region proposals which is a convolutional network, thus the superiority of GPU can be utilized.

There are three parts in Faster R-CNN. The architecture is showed in the following flow graphs. Picture_1 shows the architecture in more details, including all of the layers, softmax and so on. Picture_2 describes how does Faster R-CNN works in general.





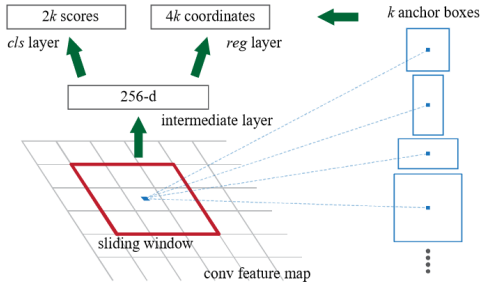
Picture_1

Picture_2

1) First, it uses CNN (including 13 convolutional layer, 13 relu layers, and 4 pooling layers) to

get each pixel of an image and return a feature map. Then the feature map will be used in both RPN and RoI pooling.

2) The RPN is the core of Faster R-CNN, it generates region proposals. First of all, it is a convolutional network and could be executed on GPU, which has a frame rate of 5fps. It has two sub-tasks, one is detecting whether the content of a box is an object or background, another task is predicting the region proposals if it is an object. The previous algorithm used the resized image or multiple filter sizes to predict the region proposal, in comparison, Faster R-CNN uses a convolutional network to predict the region proposals, which makes it faster and it can take any size image as input. RPN uses a sliding-window on the feature map generated by the CNN part and predicts multiple region proposals simultaneously at each sliding-window location. After that, Faster R-CNN uses the loss function to regress the region proposals' position. Moreover, if a region proposal's Intersection over Union (IoU) is less than 0.7, the algorithm will ignore it, this will significantly reduce the number of region proposals (from 2000 to 300) and increase their qualities.



The sliding windows of RPN

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*).$$

Loss Function

IoU: It measures the overlap between two boxes. We use that to measure how much our predicted boundary overlaps with the ground truth (the real object boundary).

$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$

3) The RoI Pooling layer collect the feature maps from CNN and region proposals from RPN, it combines the information and get proposal feature maps, and then the fully connected network uses these new maps to do multi-classification (In VOC 2007 dataset, there are 20 categories).

6.2 The reason of choosing Faster R-CNN

Object detection is currently a very hot area of computer vision research, a powerful and efficient object detection algorithm can provide tremendous helps for human's work. Also, in recent years many powerful object detection algorithms were invented, including SSD (Single Shot MultiBox Detector), R-FCN (Region-based Fully Convolutional Networks), YOLO (You Only Look Once) and many improved versions of them. Faster R-CNN is a very classic object detection algorithm, and many of its structures have been referenced by the algorithms invented in recent years. Therefore, for the purpose to get started in the area of object detection and to learn how to use TensorFlow machine learning platform, we think it is a good choice for us to start with a deep understanding of Faster R-CNN.

6.3 Details of implementation

6.3.1 Package/platform used

The packages need to be installed are:

numpy
opencv-python (the version we are using is openCV 4.1.2 + its contrib modules)
matplotlib
tensorflow_gpu (the version we are using is 1.15, please note that any version older that it cannot work with python 3.7)
keras
cython
easydict
Pillow
matplotlib
scipy

Platform:

Google Cloud Platform (the GCP, which is introduced in the part 4 of this report)

6.3.2 A brief description of the code we used

Because Faster R-CNN is a complex algorithm that is impossible for us to implement it from the beginning for this project, thus what we did is first finding an implementation of it from GitHub and make it can successfully do training and testing. Then, with the knowledge from the paper and other related materials, understand the code and write comments. After that, find the way to calculate evaluation metrics. Finally, try some modifications to the code and then compare their performances with the original one.

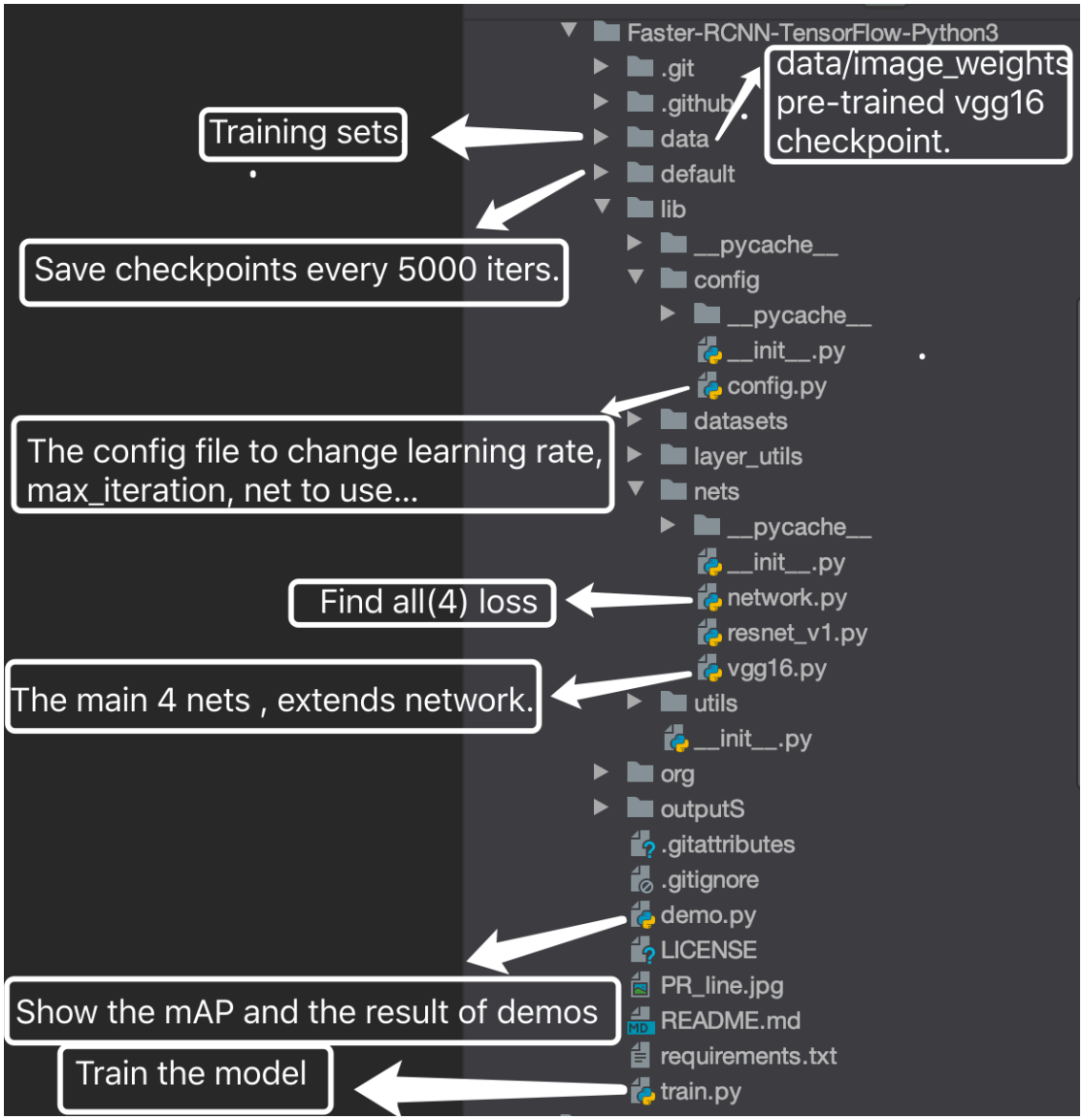
The implementation we used is from https://github.com/dBeker/Faster-RCNN-TensorFlow-Python3. We chose it because it is the highest stared TensorFlow implementation in the GitHub that can either run on Linux or Windows (although at the end we found we have not run it on

Windows yet).

6.3.2.1 The Configuration file (contains basic parameters)

CSI4106_Faster_RCNN/lib/config/config.py

6.3.2.2 A brief description of the structure of networks



/lib/nets/vgg16.py builds 4 networks:

- **build_head: The first net**
  - In VGG 16, 3x3 convolution layers would not change the size of feature map `rpn = slim.conv2d(net, 512, [3, 3], trainable=is_training, weights_initializer=initializer, scope="rpn_conv/3x3")`
  - 2x2 pooling changes the size from 2x2 to 1x1, therefore, the number of pooling layers determines the size
  - 5 conv2d layers and 4 pooling layers (0.5*0.5*0.5*0.5 = 1/16)

- It's not fully-connected, therefore, the input of the model does not have to resize to the same size since CNN can get any size of image

- **build_rpn: The second net**
  - Input is the feature maps from vgg16
  - In general, have 3x3 convolution and get 256 feature maps, at this moment, there isn't any bounding box (bbox).
  - For each point in the feature map, there is an anchor which corresponds to a region of original input(16× larger (because there are 4 pooling layers, thus $1/0.5^4 = 16$))
  - Therefore, each point (anchor) on the feature map is on a square corresponds to a size of 16 times of the original image and find k anchor boxes.
  - For a 3x3 region on conv feature map, we can find size of 3 types (1x1, 2x1 and 1x2) and 3 base sizes of anchor boxes. Therefore, k is 3x3=9 (9 anchor boxes for a point on a conv feature map).
  - Through conv layers and pooling layers, a 600*1000 image can get 256 feature maps with size about 40*60.
  - In this way, we have around 2000 boxes for each feature map. Then later we can do classification and regression on the boxes.
  - Classification Layer (2*9=18 1x1 conv)
  - For k (= 9) anchor boxes, do binary classification (foreground or background)
  - A 1x1 conv layer gets 2k scores.
  - Regression Layer (4*9=36 1x1 conv)
  - For each box, there are x1, y1, x2, y2. We regression on these points to adjust the size, shape and position of boxes.
  - Since there are k (=9) anchor boxes for each point, we have 4(x1,y1,x2,y2)*k coordinates

- **build_proposals: The third net**

  IOU = Intersaction of Unit;
  NMS = Non-maximum suppression;
  bbox = boundary box.

  - Background:
    - Input is the boxes that generated in RPN and feature map generated in vgg16.
    - At this moment, we map all boxes to original image.
    - Therefore, we can calculate IOU.
    - When IOU > a standard (0.7), it is an object.

        - Each bbox has a probility for an object, for each anchnor, we have 9 bbox with 9 probabilities (of objectiveness) in RPN layer.
    - Reduce bbox:
        - If we have 2000 boxes, it's too many. We need to reduce the number of boxes and only keep boxes that are valuable. There are three methods to reduce the boxes
            a. IOU:
            If IOU<0.7, we don't do regression. Stop.
            b. NMS:
            If it's an object (>=0.7), IOU is large. There are many boxes around the same object. Use NMS to keep only the boxes with high probabilities that says it is an object.
            c. MAX:
            If the boundary box is out of the image, ignore the bbox.
    - As a result, we reduce the number of bboxes from 2000 to 128 (sorted based on the probability of it's an object, and only take the highest 128 bboxes)

- **build_predictions: The fourth net**
    Here is a network that's fully connected, and the size of fc6, fc7 are both 4096.
    `fc6 = slim.fully_connected(pool5_flat, 4096, scope='fc6')`
    `fc7 = slim.fully_connected(fc6, 4096, scope='fc7')`
    Using scores and predictions to do classifications and bbox regression.
    - Do Classifications
        - cls_score is the result classification(21)
        `cls_score = slim.fully_connected(fc7, self._num_classes, weights_initializer=initializer, trainable=is_training, activation_fn=None, scope='cls_score')`
        - input the classification score into the fully connected layer
        `cls_prob = self._softmax_layer(cls_score, "cls_prob")`
    - Do bbox regression(21x4)
        - Already have target, proposal target layer.
        `bbox_prediction = slim.fully_connected(fc7, self._num_classes * 4, weights_initializer=initializer_bbox, trainable=is_training, activation_fn=None, scope='bbox_pred')`

6.3.2.3 The losses

/lib/nets/network.py computes 4 losses:

- loss1: RPN's binary classification (whether it's an object or background)
  `rpn_cross_entropy = tf.reduce_mean( tf.nn.sparse_softmax_cross_entropy_with_logits(logits=rpn_cls_score, labels=rpn_label))`
- loss2: RPN's loss of regression (bbox)
  `rpn_loss_box = self._smooth_l1_loss(rpn_bbox_pred, rpn_bbox_targets, rpn_bbox_inside_weights, rpn_bbox_outside_weights, sigma=sigma_rpn, dim=[1, 2, 3])`
- loss3: cross_entropy of the softmax after fully connected network (20 classes)
  `cross_entropy = tf.reduce_mean( tf.nn.sparse_softmax_cross_entropy_with_logits(logits=tf.reshape(cls_score, [-1, self._num_classes]), labels=label))`
- loss4: regression on bboxes
  `loss_box = self._smooth_l1_loss(bbox_pred, bbox_targets, bbox_inside_weights, bbox_outside_weights) loss = cross_entropy + loss_box + rpn_cross_entropy + rpn_loss_box`

6.3.2.4 How to successfully install and train the network on VOC 2007 dataset

It is obvious that code we used from GitHub cannot run directly, because there is no dataset in it. But we also need to do some modifications to the names of some folders after obtained the dataset, because their names are not consistent with the paths used in the codes.

(1) Find a new directory to store this implementation (in this example, I created a folder named 4106project)
mkdir 4106project

(2) Enter this folder
cd 4106project/

(3) Clone the implementation from GitHub
git clone https://github.com/dBeker/Faster-RCNN-TensorFlow-Python3.git

(4) Enter the root path of the implementation
cd Faster-RCNN-TensorFlow-Python3/

(5) Install the packages needed by this implementation
pip install -r requirements.txt

(6) Enter directory data/coco/PythonAPI
cd data/coco/PythonAPI

(7) Run the following two commands (if there are some warnings, you can ignore them)

python setup.py build_ext --inplace
python setup.py build_ext install

(8) Go back to the root path again and enter lib/utils
cd ..
cd ..
cd ..
cd lib/utils

(10) Run the following command
python setup.py build_ext --inplace

(11) Return back to the root path and then enter data folder
cd ..
cd ..
cd data

(12) Download PASCAL VOC 2007 dataset and extract the data
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
tar xvf VOCtrainval_06-Nov-2007.tar
tar xvf VOCtest_06-Nov-2007.tar
tar xvf VOCdevkit_08-Jun-2007.tar

(13) Create a folder named imagenet_weights to store the pre-trained model
mkdir imagenet_weights

(14) Enter this folder
cd imagenet_weights

(15) Download the vgg_16 model, extract it and delete the downloaded file to save the storage space
wget http://download.tensorflow.org/models/vgg_16_2016_08_28.tar.gz
tar -xzvf vgg_16_2016_08_28.tar.gz
rm -rf vgg_16_2016_08_28.tar.gz

(16) Change the name of the pre-trained model to be consistent with the name used in the code
mv vgg_16.ckpt vgg16.ckpt

(17) Go back to the data folder and change the name of the dataset folder to be consistent with the name used in the code
cd ..
mv VOCdevkit VOCdevkit2007

(18) Now return back to the root path and you are ready to train the model
cd ..

(19) If you want to start training, just type the following command and press “Enter” to start
(Then the model will be saved automatically every 5000 epochs, they will be saved at default/voc_2007_trainval/default/)
python train.py

The above commands could be collected together and run directly.

6.3.2.5 How to successfully run the demo.py in the root path after training

To see the result of object detection (i.e. Draw rectangular boxes to circle all of the objects in the image, and for every box give the label of the object it circles with the probability the object belongs to the label) on some demo input pictures (stored in data\demo), the demo.py in the root path will first restore the model you saved when training, then read the images stored in data\demo one by one and do the object detection process, then show all of the result images.

But there are some problems in the code, and also showing the result images directly is a little bit tricky for us when using the GCP (because to show the images using plt.show(), the result images must be downloaded to the local memory and then be showed, but the download speed is very slow if we do not use the university’s Internet, and we cannot see these images again after we closed the windows showing these images) so we modified the demo.py to make it works and can store the result images to a folder assigned by the user (input the folder name in the command window), we also added some code to implement the functionality of calculating the evaluation metrics on test set, thus after running the demo.py, you can also see the average detection time of testing all of the images in the test set, and the APs (Average Precisions) of each class and the mAP (mean Average Precision). What’s more, we also added some code in /lib/datasets/pascal_voc.py to draw the PR line of the model that is testing, the picture of the PR line will be stored as PR_line.jpg in the root path.

To use the demo.py we modified, you can easily find it in the root path of our GitHub repository, then replace the original demo.py in the root path by our modified demo.py. Also, you need to replace another two files pascal_voc.py and test.py by our modified version stored in the same path in our GitHub repository as they are in your project.

So you need to replace three files:
/demo.py
/lib/utils/test.py
/lib/datasets/pascal_voc.py

Then in the root path, type python demo.py in your command window to use it.

The following is an example of using our modified demo.py:
First it will do object detection on the 4952 images in the test set, in each row, the second last time is the time used for doing object detection on the image it is processing, the last time is the time used for comparing the detection result with the true result of the test image.

```
Loaded network default/voc_2007_trainval/default/vgg16_faster_rcnn_iter_50000.ckpt
2019-12-08 01:06:53.494423: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcubl
as.so.10.0
2019-12-08 01:06:57.595830: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudn
n.so.7
im_detect: 1/4952 21.127s 0.001s
im_detect: 2/4952 10.905s 0.001s
im_detect: 3/4952 7.477s 0.001s
im_detect: 4/4952 5.805s 0.001s
im_detect: 5/4952 4.651s 0.001s
im_detect: 6/4952 3.882s 0.001s
im_detect: 7/4952 3.418s 0.001s
im_detect: 8/4952 3.079s 0.001s
im_detect: 9/4952 2.741s 0.001s
im_detect: 10/4952 2.537s 0.001s
im_detect: 11/4952 2.309s 0.001s
im_detect: 12/4952 2.120s 0.001s
im_detect: 13/4952 1.981s 0.001s
im_detect: 14/4952 1.842s 0.001s
im_detect: 15/4952 1.757s 0.001s
im_detect: 16/4952 1.686s 0.001s
im_detect: 17/4952 1.624s 0.001s
im_detect: 18/4952 1.572s 0.001s
im_detect: 19/4952 1.491s 0.001s
im_detect: 20/4952 1.418s 0.001s
im_detect: 21/4952 1.352s 0.001s
im_detect: 22/4952 1.322s 0.001s
im_detect: 23/4952 1.267s 0.001s
im_detect: 24/4952 1.242s 0.001s
im_detect: 25/4952 1.193s 0.001s
im_detect: 26/4952 1.149s 0.001s
im_detect: 27/4952 1.108s 0.001s
im_detect: 28/4952 1.085s 0.001s
im_detect: 29/4952 1.049s 0.001s
im_detect: 30/4952 1.015s 0.001s
im_detect: 31/4952 0.984s 0.001s
im_detect: 32/4952 0.954s 0.001s
im_detect: 33/4952 0.947s 0.001s
im_detect: 34/4952 0.920s 0.001s
im_detect: 35/4952 0.895s 0.001s
im_detect: 36/4952 0.871s 0.001s
im_detect: 37/4952 0.849s 0.001s
im_detect: 38/4952 0.827s 0.001s
```

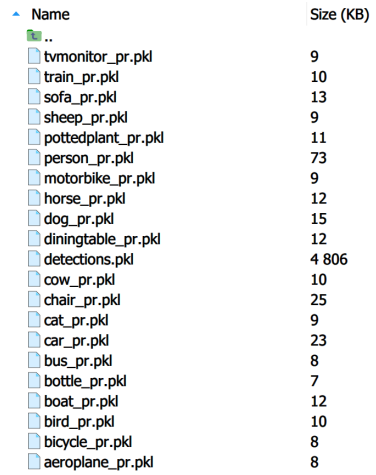
After tested on all of the images in the test set, it will calculate and output the average detection time.

```
im_detect: 4944/4952 0.074s 0.001s
im_detect: 4945/4952 0.074s 0.001s
im_detect: 4946/4952 0.074s 0.001s
im_detect: 4947/4952 0.074s 0.001s
im_detect: 4948/4952 0.074s 0.001s
im_detect: 4949/4952 0.074s 0.001s
im_detect: 4950/4952 0.074s 0.001s
im_detect: 4951/4952 0.074s 0.001s
im_detect: 4952/4952 0.074s 0.001s


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

The average detection time over 4952 images is 0.135915028567338

Evaluating detections

Writing aeroplane VOC results file
Writing bicycle VOC results file
Writing bird VOC results file
```

Then it will write out the information of detection result of each class to directory /default/voc_2007_test/default/, and output the APs for each class and the mAP.

```
Evaluating detections

Writing aeroplane VOC results file
Writing bicycle VOC results file
Writing bird VOC results file
Writing boat VOC results file
Writing bottle VOC results file
Writing bus VOC results file
Writing car VOC results file
Writing cat VOC results file
Writing chair VOC results file
Writing cow VOC results file
Writing diningtable VOC results file
Writing dog VOC results file
Writing horse VOC results file
Writing motorbike VOC results file
Writing person VOC results file
Writing pottedplant VOC results file
Writing sheep VOC results file
Writing sofa VOC results file
Writing train VOC results file
Writing tvmonitor VOC results file
VOC07 metric? Yes
AP for aeroplane = 0.7005
AP for bicycle = 0.7075
AP for bird = 0.5869
AP for boat = 0.5559
AP for bottle = 0.4143
AP for bus = 0.6777
AP for car = 0.7139
AP for cat = 0.7084
AP for chair = 0.4341
AP for cow = 0.7337
AP for diningtable = 0.6347
AP for dog = 0.7481
AP for horse = 0.7998
AP for motorbike = 0.6782
AP for person = 0.7029
AP for pottedplant = 0.3167
AP for sheep = 0.6586
AP for sofa = 0.6368
AP for train = 0.7476
AP for tvmonitor = 0.5035
Mean AP = 0.6330
```

The output information of detection result will be stored as .pkl files, as the following shows:



Name	Size (KB)
..	
tvmonitor_pr.pkl	9
train_pr.pkl	10
sofa_pr.pkl	13
sheep_pr.pkl	9
pottedplant_pr.pkl	11
person_pr.pkl	73
motorbike_pr.pkl	9
horse_pr.pkl	12
dog_pr.pkl	15
diningtable_pr.pkl	12
detections.pkl	4 806
cow_pr.pkl	10
chair_pr.pkl	25
cat_pr.pkl	9
car_pr.pkl	23
bus_pr.pkl	8
bottle_pr.pkl	7
boat_pr.pkl	12
bird_pr.pkl	10
bicycle_pr.pkl	8
aeroplane_pr.pkl	8

You may notice that there is a sentence

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

---

at the end of the output, because we invoked the method of calculating the APs and the mAP in pascal_voc.py. We did tried to use reval.py, but this file is not contained in this version of Faster R-CNN in the GitHub, actually this file is used by the Caffe version of Faster R-CNN in another GitHub repository. Then we have studied how to use the reval.py in our version, but unfortunately the structure of the input it required is very different from the version of Faster R-

CNN we are using. And considering it says “Results should be very close to the official MATLAB eval code” and we are not calculate the result for a paper, we think we can accept the output result for our evaluation process.

```
AP for motorbike = 0.6782
AP for person = 0.7029
AP for pottedplant = 0.3167
AP for sheep = 0.6586
AP for sofa = 0.6368
AP for train = 0.7476
AP for tvmonitor = 0.5035
Mean AP = 0.6330
~~~~~~~~
Results:
0.700
0.707
0.587
0.556
0.414
0.678
0.714
0.708
0.434
0.734
0.635
0.748
0.800
0.678
0.703
0.317
0.659
0.637
0.748
0.504
0.633
~~~~~~~~

--------------------------------------------------------------
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
--------------------------------------------------------------
============================================================================

Please input the folder name you want to output the result images:
```

Now you can input the folder name you want to output the result images that added boxes along with the probability values, input the name (please just input the name but not a path of directory) and press “Enter”, you will see the following:
(As an example, we input “op1”)

```
~~~~~~~~

--------------------------------------------------------------
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
--------------------------------------------------------------
============================================================================

Please input the folder name you want to output the result images:

QXcbConnection: XCB error: 145 (Unknown), sequence: 171, resource id: 0, major code: 139 (Unknown), minor code: 20
op1
Directory: op1 successfully created
----------------------------------------------------------------------------
Demo for data/demo/000456.jpg
Detection took 0.041s for 300 object proposals
----------------------------------------------------------------------------
Demo for data/demo/000457.jpg
Detection took 0.042s for 300 object proposals
----------------------------------------------------------------------------
Demo for data/demo/000542.jpg
Detection took 0.042s for 300 object proposals
----------------------------------------------------------------------------
Demo for data/demo/001150.jpg
Detection took 0.043s for 300 object proposals
----------------------------------------------------------------------------
Demo for data/demo/001763.jpg
Detection took 0.041s for 300 object proposals
----------------------------------------------------------------------------
Demo for data/demo/004545.jpg
Detection took 0.043s for 300 object proposals
(base) accelerator@vaccelerator3:~/csi4106/faster_default_50000/Faster-RCNN-TensorFlow-Python3$
```

(Sometimes it will output an error information about QXcbConnection, but it has no impact to the output results)
This is the end of running demo.py, now all of the output images are stored in a folder you named just now in the root path, as the following shows:

This is the folder op1 in the root path:

Name	Size (KB)
..	
000456.jpg_0.jpg	6 052
000456.jpg_1.jpg	6 031
000456.jpg_2.jpg	6 064
000457.jpg_0.jpg	6 052
000457.jpg_1.jpg	6 031
000457.jpg_2.jpg	6 064
000542.jpg_0.jpg	6 910
001150.jpg_0.jpg	5 191
001150.jpg_1.jpg	5 250
001763.jpg_0.jpg	4 666
001763.jpg_1.jpg	4 664
004545.jpg_0.jpg	6 976
004545.jpg_1.jpg	6 986
004545.jpg_2.jpg	7 013
004545.jpg_3.jpg	7 022

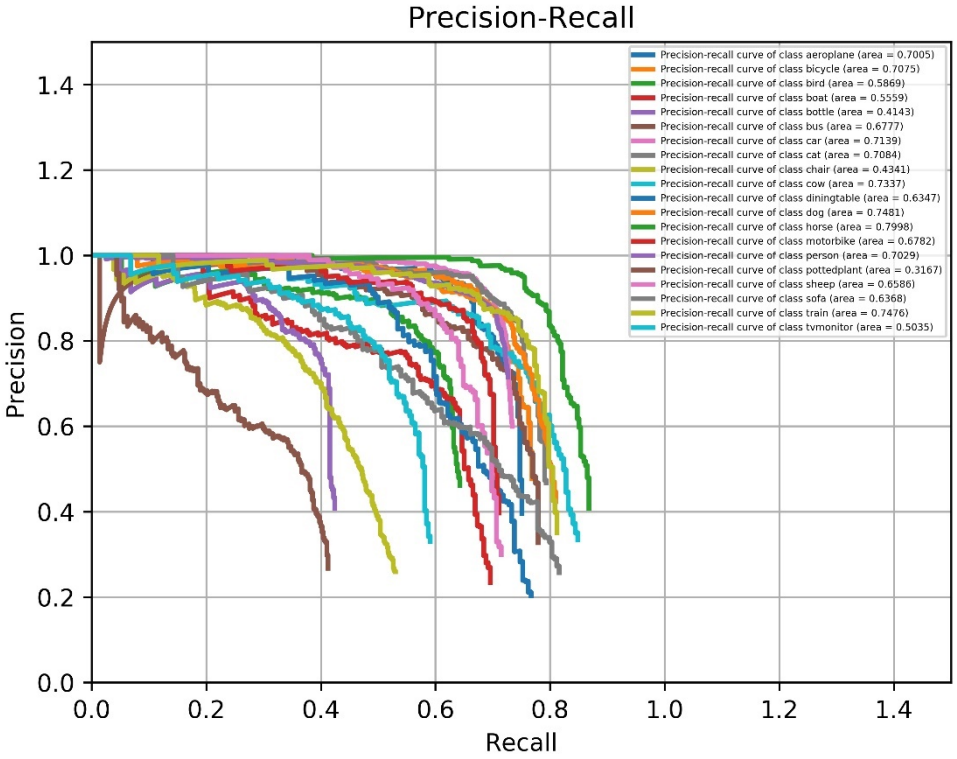
For every image, the program generates several images end with different number, each of them is the detection result for one single class. In the above example, the program detects 4 classes of objects in 004545.jpg, then the program will generate 4 images for it, as the following shows:

Original image:

Outputs:

The program can also generate the PR line for each class and store it in the root path of the project:

Name	Size (KB)
..	
.git	
.github	
data	
default	
lib	
op1	
output006	
.gitattributes	1
.gitignore	1
demo.py	7
LICENSE	1
PR_line.jpg	1 062
README.md	2
requirements.txt	1
train.py	7



6.3.3 Files we have commented

Assume the root path of our project is CSI4106_Faster_RCNN:

1. CSI4106_Faster_RCNN/lib/nets/vgg16.py
2. CSI4106_Faster_RCNN/train.py
3. CSI4106_Faster_RCNN/demo.py
4. CSI4106_Faster_RCNN/lib/nets/network.py
5. CSI4106_Faster_RCNN/lib/datasets/pascal_voc.py
6. CSI4106_Faster_RCNN/lib/layer_utils/proposal_layer.py
7. CSI4106_Faster_RCNN/lib/utils/bbox_transform.py
8. CSI4106_Faster_RCNN/lib/datasets/factory.py
9. CSI4106_Faster_RCNN/lib/utils/test.py

6.3.4 Our modified versions

We have modified the code of the implementation of Faster R-CNN to have a better understanding to the code and the influence of different modifications to the performance. However, because the implementation of Faster R-CNN is very complex, after some modifications, the code cannot run successfully or the result model does not work, and we do not have time to find out the reason behind them, but we still write down our efforts here.

By the way, we have modified the max number of epochs from the default 40000 epochs to 50000 epochs for all of the versions we modified (except the ones mentioned specially), to make the performance more significant for comparing.

6.3.4.1 Modifications to the batch size

This is the easiest modification, to modify the batch size, one can simply change the value of "batch_size" in the configuration file config.py stored in /lib/config/.

```
21  ######################
22  # Training Parameters #
23  ######################
24  tf.app.flags.DEFINE_float('weight_decay', 0.0005, "Weight decay, for regularization")
25  tf.app.flags.DEFINE_float('learning_rate', 0.001, "Learning rate")
26  tf.app.flags.DEFINE_float('momentum', 0.9, "Momentum")
27  tf.app.flags.DEFINE_float('gamma', 0.1, "Factor for reducing the learning rate")
28
29  tf.app.flags.DEFINE_integer('batch_size', 256, "Network batch size during training")
30  tf.app.flags.DEFINE_integer('max_iters', 50000, "Max iteration")
31  tf.app.flags.DEFINE_integer('step_size', 30000, "Step size for reducing the learning rate, currently only support one step")
32  tf.app.flags.DEFINE_integer('display', 10, "Iteration intervals for showing the loss during training, on command line interface")
33
34  tf.app.flags.DEFINE_string('initializer', "truncated", "Network initialization parameters")
35  tf.app.flags.DEFINE_string('pretrained_model', "./data/imagenet_weights/vgg16.ckpt", "Pretrained network weights")
36
37  tf.app.flags.DEFINE_boolean('bias_decay', False, "Whether to have weight decay on bias as well")
38  tf.app.flags.DEFINE_boolean('double_bias', True, "Whether to double the learning rate for bias")
39  tf.app.flags.DEFINE_boolean('use_all_gt', True, "Whether to use all ground truth bounding boxes for training, "
40                                                  "For COCO, setting USE_ALL_GT to False will exclude boxes that are flagged as ''is
41  tf.app.flags.DEFINE_integer('max_size', 1000, "Max pixel size of the longest side of a scaled input image")
42  tf.app.flags.DEFINE_integer('test_max_size', 1000, "Max pixel size of the longest side of a scaled input image")
```

As you can see, the default value is 256. We have tried different values 64, 128, 256, 512 and 1024 and trained all of them for the comparison in result analysis phase.

And in this file of all of the versions we modified (except the ones mentioned specially), we modified the "max_iters" from the default 40000 to 50000 to increase the number of training epochs, to make the performance more significant for comparing.

6.3.4.2 Modifications to the CNN structure used in the algorithm

The default CNN structure used in the original implementation is VGG 16, it uses /lib/nets/vgg16.py to define the structure of the CNN part it uses and downloaded the pretrained model from http://download.tensorflow.org/models/vgg_16_2016_08_28.tar.gz.

To find more CNN structures with available pretrained models, we found a model library: https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-models, it has many CNN codes that defines its structure and each one has a well trained model with it, as the following screenshot shows:

Model	TF-Slim File	Checkpoint	Top-1 Accuracy	Top-5 Accuracy
Inception V1	Code	inception_v1_2016_08_28.tar.gz	69.8	89.6
Inception V2	Code	inception_v2_2016_08_28.tar.gz	73.9	91.8
Inception V3	Code	inception_v3_2016_08_28.tar.gz	78.0	93.9
Inception V4	Code	inception_v4_2016_09_09.tar.gz	80.2	95.2
Inception-ResNet-v2	Code	inception_resnet_v2_2016_08_30.tar.gz	80.4	95.3
ResNet V1 50	Code	resnet_v1_50_2016_08_28.tar.gz	75.2	92.2
ResNet V1 101	Code	resnet_v1_101_2016_08_28.tar.gz	76.4	92.9
ResNet V1 152	Code	resnet_v1_152_2016_08_28.tar.gz	76.8	93.2
ResNet V2 50^	Code	resnet_v2_50_2017_04_14.tar.gz	75.6	92.8
ResNet V2 101^	Code	resnet_v2_101_2017_04_14.tar.gz	77.0	93.7
ResNet V2 152^	Code	resnet_v2_152_2017_04_14.tar.gz	77.8	94.1
ResNet V2 200	Code	TBA	79.9*	95.2*
VGG 16	Code	vgg_16_2016_08_28.tar.gz	71.5	89.8
VGG 19	Code	vgg_19_2016_08_28.tar.gz	71.1	89.8
MobileNet_v1_1.0_224	Code	mobilenet_v1_1.0_224.tgz	70.9	89.9
MobileNet_v1_0.50_160	Code	mobilenet_v1_0.50_160.tgz	59.1	81.9
MobileNet_v1_0.25_128	Code	mobilenet_v1_0.25_128.tgz	41.5	66.3
MobileNet_v2_1.4_224^*	Code	mobilenet_v2_1.4_224.tgz	74.9	92.5

After compared the code in /lib/nets/vgg16.py and the code in the above website, we found that the author used the same CNN code as the part before the fully connected layer of VGG 16's in the above website. So we think maybe it is possible to try some other CNN structures with the similar replacement. We have tried two variations of modification: changing the CNN structure to VGG 19 and ResNet V1 152.

To change the CNN structure to VGG 19, we know that VGG 16 has 16 layers and VGG19 has 19 layers (3 more conv layers in total), but it is not successful if we simply add the 3 more layers, because the size of VGG 19 pretrained model does not match the modified structure. After a hard study of CNN, TensorFlow and the code we are using, one of our members changed the conv3's conv layers from 'conv3/conv3_1', 'conv3/conv3_2' and 'conv3/conv3_3' to 'conv3/conv3_1', 'conv3/conv3_2' ,'conv3/conv3_3', and 'conv3/conv3_4'. Repeat the same procedure to conv4 and conv5. Eventually, the structure matches the pretrained data of VGG19.

We have also tried to change the CNN part to ResNet V1-152, we found there is a resnet_v1.py file in the same folder as vgg16.py's, we modified the configuration file, tried to changed the structure of block to fit the network, then we were able to change the blocks and generate the suitable network. But a problem of Value Error (as the following screenshot shows) was hard to fix, we searched about the error and found the problem is most likely caused by the inconsistency of the version of a package. But this error was still hard to solve after several tries, the version problem is still there. Finally, we believe we should not spend more time on

the ResNet and just record our effort on it.

```
Traceback (most recent call last):
  File "train.py", line 218, in <module>
    train.train()
  File "train.py", line 127, in train
    restorer = tf.train.Saver(variables_to_restore)
  File "/home/kevin/anaconda3/lib/python3.7/site-packages/tensorflow_core/python/training/saver.py", line 828, in __
init__
    self.build()
  File "/home/kevin/anaconda3/lib/python3.7/site-packages/tensorflow_core/python/training/saver.py", line 840, in bu
ild
    self._build(self._filename, build_save=True, build_restore=True)
  File "/home/kevin/anaconda3/lib/python3.7/site-packages/tensorflow_core/python/training/saver.py", line 865, in _b
uild
    raise ValueError("No variables to save")
ValueError: No variables to save
```

6.3.4.3 Modifications to the learning rate and the max number of training epochs

We can simply modify the learning rate and the max number of training epochs by simply change the 'learning_rate' and 'max_iters' in the configuration file config.py stored in /lib/config/ respectively.

We have tried to reduce the learning rate to half of the original one, i.e., 0.0005 and increase the max number of training epochs to 150000, then trained the model to compare with others in the result analysis phase.

6.3.4.4 Modifications to the optimizer

We have tried to modify the following part to what the following screenshot shows to change the optimizer from MomentumOptimizer to AdamOptimizer.

```
77          # Create session
78          tfconfig = tf.ConfigProto(allow_soft_placement=True)
79          tfconfig.gpu_options.allow_growth = True
80          sess = tf.Session(config=tfconfig)
81
82          with sess.graph.as_default():
83
84              tf.set_random_seed(cfg.FLAGS.rng_seed)
85              layers = self.net.create_architecture(sess, "TRAIN", self.imdb.num_classes, tag='default')
86              loss = layers['total_loss']
87
88
89              # CSI 4106: changed the optimizer to AdamOptimizer
90              lr = tf.Variable(cfg.FLAGS.learning_rate, trainable=False)
91              # momentum = cfg.FLAGS.momentum
92              # optimizer = tf.train.MomentumOptimizer(lr, momentum)
93
94              optimizer = tf.train.AdamOptimizer(learning_rate = lr)
95              gvs = optimizer.compute_gradients(loss)
96
97              # Double bias
98              # Double the gradient of the bias if set
99              if cfg.FLAGS.double_bias:
100                 final_gvs = []
101                 with tf.variable_scope('Gradient_Mult'):
102                     for grad, var in gvs:
103                         scale = 1.
104                         if cfg.FLAGS.double_bias and '/biases:' in var.name:
105                             scale *= 2.
106                         if not np.allclose(scale, 1.0):
107                             grad = tf.multiply(grad, scale)
108                         final_gvs.append((grad, var))
109                 train_op = optimizer.apply_gradients(final_gvs)
110             else:
111                 train_op = optimizer.apply_gradients(gvs)
```

After that, the code can successfully run the training and during training the loss reduces as other successful variations, but when we test the result using demo.py, it is strange that the APs and mAP are all zero, as the following shows:

```
AP for aeroplane = 0.0000
AP for bicycle = 0.0000
AP for bird = 0.0000
AP for boat = 0.0000
AP for bottle = 0.0000
AP for bus = 0.0000
AP for car = 0.0000
AP for cat = 0.0000
AP for chair = 0.0000
AP for cow = 0.0000
AP for diningtable = 0.0000
AP for dog = 0.0000
AP for horse = 0.0000
AP for motorbike = 0.0000
AP for person = 0.0000
AP for pottedplant = 0.0000
AP for sheep = 0.0000
AP for sofa = 0.0000
AP for train = 0.0000
AP for tvmonitor = 0.0000
Mean AP = 0.0000
~~~~~~~~
Results:
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
0.000
```

And there is no output in the output folder for the demo images, we have studied it for a while but there is no clue that shows what caused the problem, considering we do not have enough time, we just record our effort on it

6.3.4.4 Modifications to the minibatch

We have also tried to modify the number of images used in every minibatch, we modified 'ims_per_batch' in the configuration file, but files /lib/utils/minibatch.py seems does not want us to change this number, it uses “assert” to fix that number to 1:

```python
def get_minibatch(roidb, num_classes):
    """Given a roidb, construct a minibatch sampled from it."""
    num_images = len(roidb)
    # Sample random scales to use for each image in this batch
    random_scale_inds = npr.randint(0, high=len(cfg.FLAGS2["scales"]),
                                    size=num_images)
    assert (cfg.FLAGS.batch_size % num_images == 0), 'num_images ({}) must divide BATCH_SIZE ({})'.format(num_images, cfg.FLAGS.batch_size)

    # Get the input image blob, formatted for caffe
    im_blob, im_scales = _get_image_blob(roidb, random_scale_inds)

    blobs = {'data': im_blob}

    assert len(im_scales) == 1, "Single batch only"
    assert len(roidb) == 1, "Single batch only"

    # gt boxes: (x1, y1, x2, y2, cls)
    if cfg.FLAGS.use_all_gt:
        # Include all ground truth boxes
        gt_inds = np.where(roidb[0]['gt_classes'] != 0)[0]
    else:
        # For the COCO ground truth boxes, exclude the ones that are ''iscrowd''
        gt_inds = np.where(roidb[0]['gt_classes'] != 0 & np.all(roidb[0]['gt_overlaps'].toarray() > -1.0, axis=1))[0]
    gt_boxes = np.empty((len(gt_inds), 5), dtype=np.float32)
```

We have tried to delete the assertion and did many modifications to other files, but at the end we still cannot make the code works.

7. Result Analysis

7.1 Evaluation methods

As we have mentioned in the part 6.3.2.5, our evaluation metrics include the average detection speed on all of the images in the test set, the AP (Average Precisions) of each class and the mAP (mean Average Precision). What's more, we will also compare some of the output images from the same image to visually compare the result.

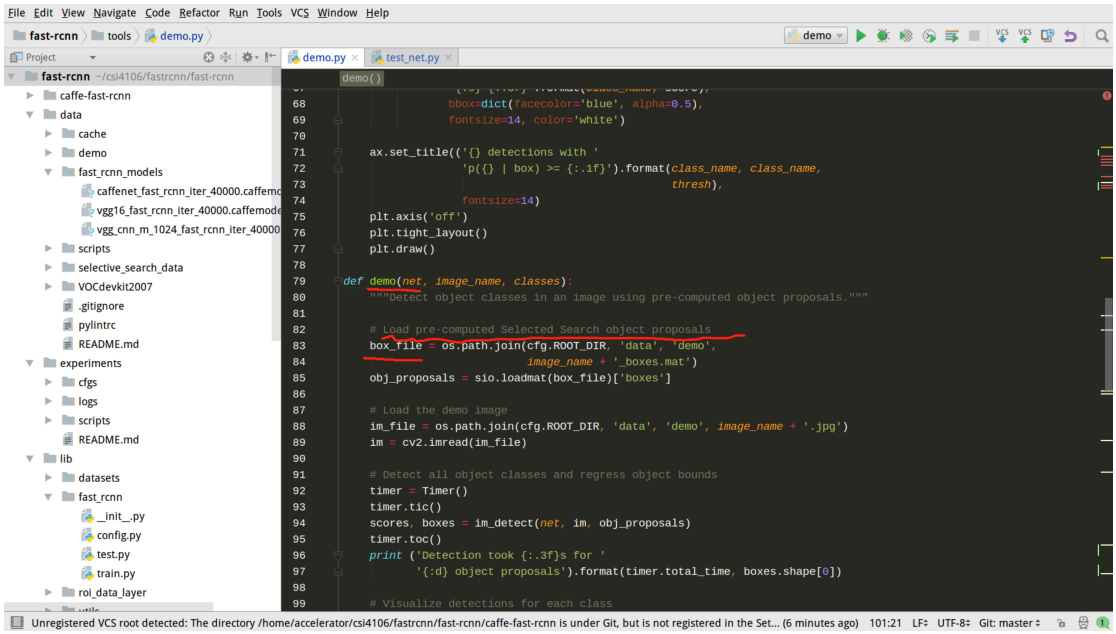
Because for a classifier, the precision and recall of it are usually grows in the opposite way, when the precision increases, usually the recall will decrease. An extreme example is if all samples are labeled as positive, then the precision will be very small, however, recall value will reach the maximum at the same time, which is 1. Thus, in this case, the precision and recall values cannot show the performance of the model well, but AP (Average Precisions) can consider these two values simultaneously.

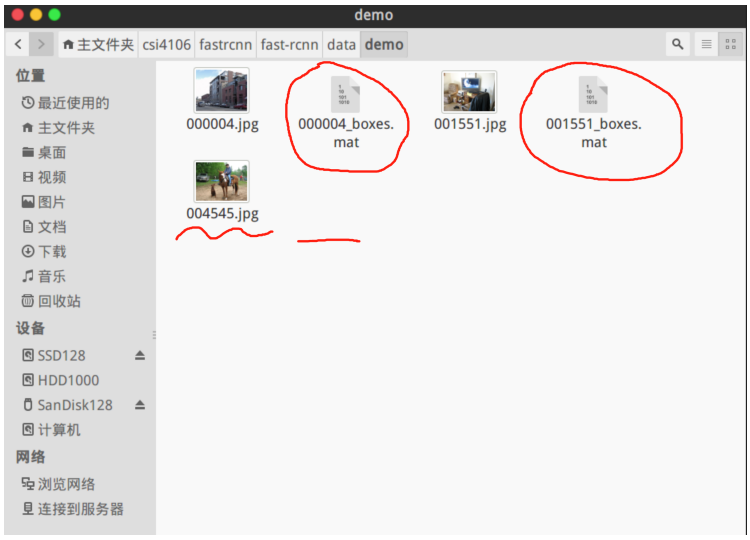
The AP for each class is calculated by the following way (this is the way to calculate AP after the PASCAL VOC CHALLENGE in 2010, although we are using VOC 2007 dataset, but this way of calculating AP will make the result more precise):
If there are M positive examples in this class in the test set, we first calculate M recall values (from 1/M, 2/M, …, to M-1/M, M/M), then for each of the recall values r_M, find the greatest precision p_M when we want to keep the recall greater than r_M, if we plot a graph for every $x = r_M$ and $y = p_M$, we can get a P-R line for this class. After that, the average of the p_Ms is the AP we get for this class, and the mAP is the mean value of the APs over all of the classes. Thus, in the graph of PR lines, the bigger the area under the PR line, the greater AP that class has.

To evaluate the speed of detection, we simply calculate the average time used in the detection process when processing each of the samples in the test set.

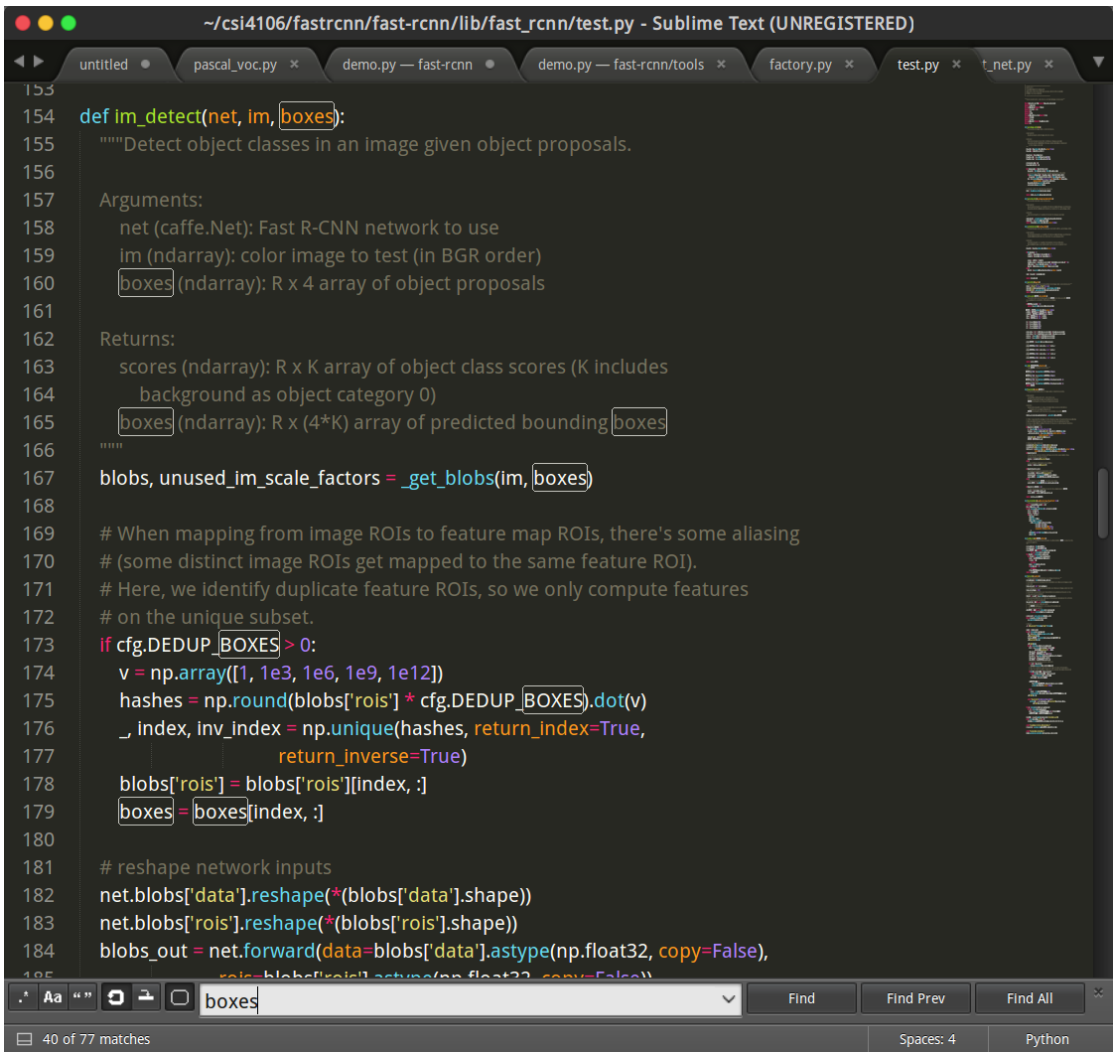
7.1.1 Comparative results

Because we did not figure out how to compute the above metrics using the baseline, and it is even a must that to test an image of our choice, the image must has a pre-computed Selected Search object proposals with it (as the following shows), which makes it is impossible to have a visual comparison with the Faster R-CNN using the same image (we are using 004545.jpg)





The following code shows that it is impossible to invoke the method im_detect without Selected Search object proposals, I guess maybe we can find some codes to calculate the Selected Search object proposals for any image we want to test, but we do not have time to discover and implement this task.



So unfortunately, we can only compare the results between our different variations of Faster R-CNN. We only compare the model trained for 50000 epochs for each of our variations.

We compare the following variations:
64: Only changed the batch_size from 256 to 64 based on the original version
128: Only changed the batch_size to 128 based on the original version
256: The original version, we did not change anything
512: Only changed the batch_size to 512 based on the original version
1024: Only changed the batch_size to 1024 based on the original version
VGG_19: Only replaced the CNN part with using the structure of VGG_19 instead of VGG_16
Half_50000: Only changed the learning rate from 0.001 to 0.0005
Half_150000: Changed the learning rate from 0.001 to 0.0005, and trained for 150000 epochs

The following table shows the comparison of average detection speed and the mAP.

	Average detection time	mAP
64	0.10273261353473803	0.6012
128	0.1044866292943799	0.6221
256	0.1032320766637447	0.6330
512	0.10242221420028526	0.6308
1024	0.102486750416565	0.5986
VGG_19	0.11066207139725968	0.3724
Half_50000	0.10425427696266554	0.6314
Half_150000	0.2014774918229321	0.6421

As we can see, higher or lower batch size does not provide better performance in mAP, the further from the original one (256), the worse the mAP will be. The performance of our modified version VGG_19 seems not good, maybe more things related to what we have changed for VGG_19 should also be changed, but unfortunately we do not have time to do more things to improve it. Comparing the mAP of 256 with Half_50000's, we can know that using lower learning rate with the same training epochs does not make the performance better. With the comparison between Half_50000 and Half_150000, we can find that more epochs of training does provide us better performance, but the average detection time of Half_150000 increases to 2 times of the others, but we have not figure out the reason yet. There is no obvious difference between the average detection time of other models.

To visually compare the result, we compare the result images outputted from demo image 004545.jpg of model 256, 1024, VGG_19 and Half_150000.

For the dog in the original image, the outputs of the four models are:
256:

1024:

VGG_19:
There is no box on the dog.

Half_150000:

We can find that the models with higher mAP indeed perform well. VGG_19 has a much lower mAP than others' and fails to find a dog from the image, 1024 does find a dog but it is not very sure that it is a dog because it gives the label of dog box a low probability value and also gives the dog a label of cow. We can also conclude that a small difference in mAP will bring a relatively big difference to the result, the mAPs for 1024, 256 and Half_150000 are 0.5986, 0.6330 and 0.6421 respectively, but 1024's performance is already visually worse than the other twos, and 256 only gives the dog it detected a probability of 0.795, compared with 1024's probability of 0.916, considering the difference of mAPs between 256 and 1024 is only 0.0091.

Also, comparing the results for the people in the original image:
256:

1024:

VGG_19:

Half_150000:

Again, VGG_19 makes a mistake, it identifies the rear of the horse as a person. All other models correctly draw the boxes, but still, the higher the mAP a model has, the higher it believes the boxes it draws are correct (as you can see, the probabilities along with the boxes increases if the model's mAP increases).

8. Conclusion

Because we did not figure out how to compute the evaluation metrics and run the image we want to test for demo using the baseline, so we can only compare the results between our different variations of Faster R-CNN. The only comparison between the Faster R-CNN and the baseline is we can roughly compare the detection time. In the demo of baseline, it takes around 0.31 second for the detection of an image, whereas it takes around 0.042 second for the demo of Faster R-CNN to complete the detection process for an image, which proved the greatest change in Faster R-CNN, i.e., generating region proposals using RPN instead of Selective search, which dramatically reduces the detection time and breaks the bottleneck of Fast R-CNN.

When we compare the results between our different variations of Faster R-CNN, we can conclude that:
(1) Higher or lower batch size does not provide obvious influence for the performance in mAP.
(2) More epochs of training do provide us better performance.
(3) There is no obvious difference between the average detection time in most cases.
(4) mAP is an important and relatively reliable evaluation metric, a small difference in mAP can indeed reflect an obvious difference in the detection result.

What's more, we would like to say Faster R-CNN is really an amazing invention and a good start for us in the area of object detection, we indeed learned a huge amount of things in this project, including learned the knowledge of CNN and Faster R-CNN, greatly improved the skill

of analyzing codes and solving problems, and being much more familiar with TensorFlow and the use of remote Linux system, etc. The code of this implementation of Faster R-CNN assembled many functionalities and considered scalability, which makes the code become very complex, but this also improved our abilities of reading code and the style of code. Also, we find that we have to face some more problems due to the nature of deep learning, the most obvious problem is it is very time consuming to find and fix some problems, some times after our modification, the code works for training, but if we find the model after training performs very poor, after fixed the code we will need to spend another 3~4 hours to wait for the training process to complete to see the performance of new model. Finally, it is really a pity that we could do more things to make this project more amazing, including making the baseline model comparable with the implementation of Faster R-CNN, however, reading paper, learning the principles of CNN and how to use TensorFlow, setting up the platform and making the code works cost us a lot of time, but it worth it. With the experience in this project, we believe we can do more amazing things in the future.

Reference:

Paper:
Shaoqing Ren, Kaiming He, Ross Girshick, & Jian Sun. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6), 1137-1149.

Code:
Faster R-CNN:
https://github.com/dBeker/Faster-RCNN-TensorFlow-Python3

Fast R-CNN (Baseline approach):
https://github.com/rbgirshick/fast-rcnn

PASCAL VOC 2007 dataset:
http://host.robots.ox.ac.uk/pascal/VOC/voc2007/

caffe-fast-rcnn:
https://github.com/rbgirshick/py-faster-rcnn

Caffe:
https://github.com/BVLC/caffe.git

A model library:
https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-models

The VGG 16 pretrained model we used:
http://download.tensorflow.org/models/vgg_16_2016_08_28.tar.gz