Multi-class Object Detetion with Deep Learning

Mingxuan Chai, Wentao Zhang, Shengkai Ni

January 2016

Contents

1 Introduction **3**

2 RCNN and SPP-net **4**
- 2.1 Main process of R-CNN 4
  - 2.1.1 Region proposals 4
  - 2.1.2 Feature extraction 4
  - 2.1.3 Category classification 4
- 2.2 R-CNN has notable drawbacks 4
- 2.3 SPPnet 6
- 2.4 SPPnet also has notable drawbacks 6

3 Fast R-CNN **7**
- 3.1 Advantages 7
- 3.2 Improvement of design 7
- 3.3 Main architecture 7

4 Our Works **8**
- 4.1 Brief introduction of directory structure 8
- 4.2 Implementation 9
  - 4.2.1 Region Proposals 9
  - 4.2.2 Training 11
  - 4.2.3 Data Test 11

5 Summary **12**

6 Result **13**
- 6.1 Feedback from MLG Lab 13
  - 6.1.1 Detection Time 13
  - 6.1.2 Proposal Time 13
  - 6.1.3 Total 13
- 6.2 Conclusion from Case Study 13

7 Future Work **15**

8 Reference **16**

1 Introduction

In our final project, we implement an object detection function based on deep learning with R-CNN. We have tested it on four datasets taken from PKU's different sites and got quite satisfactory results.

Meanwhile, we tried to work it out on Faster-RCNN. It's a pity that we do not succeed to get the final result till now.

The following introduction is about several algorithms relating to fast-RCNN and our implementing work. Then we will make a conclusion considering our test result and summarize what we learned through the project.

2 RCNN and SPP-net

This part will introduce the algorithm of RCNN and SPP-net.

2.1 Main process of R-CNN

2.1.1 Region proposals

A variety of methods can be used to generate category-independent region proposals. Examples include: sliding window, selective search, etc.

2.1.2 Feature extraction

In [1], they extract a 4096-dimensional feature vector from each region proposal using the Caffe [3] implementation of the CNN described by Krizhevsky et al. [4]. Features are computed by forward propagating a mean-subtracted 227×227 RGB image through five convolutional layers and two fully connected layers.

2.1.3 Category classification

Train a detection classifier rather than simply use outputs from the final layer (fc8) of the fine-tuned CNN. In [1], the author optimize one linear SVM per class.

2.2 R-CNN has notable drawbacks

Training is a multi-stage pipeline. R-CNN first finetunes a ConvNet on object proposals using log loss. Then, it fits SVMs to ConvNet features. These SVMs act as object detectors, replacing the softmax classifier learnt by fine-tuning. In the third training stage, bounding-box regressors are learned.

Training is expensive in space and time. With very deep networks, such as VGG16, this process takes 2.5 GPU-days for the 5k images of the VOC07 trainval set. These features require hundreds of gigabytes of storage.

Object detection is slow. At test-time, features are extracted from each object proposal in each test image. Detection with VGG16 takes 47s / image (on a GPU).

After resizing the image to 227×227, we will lose some information about the objection. The following pictures show an example of information loss.

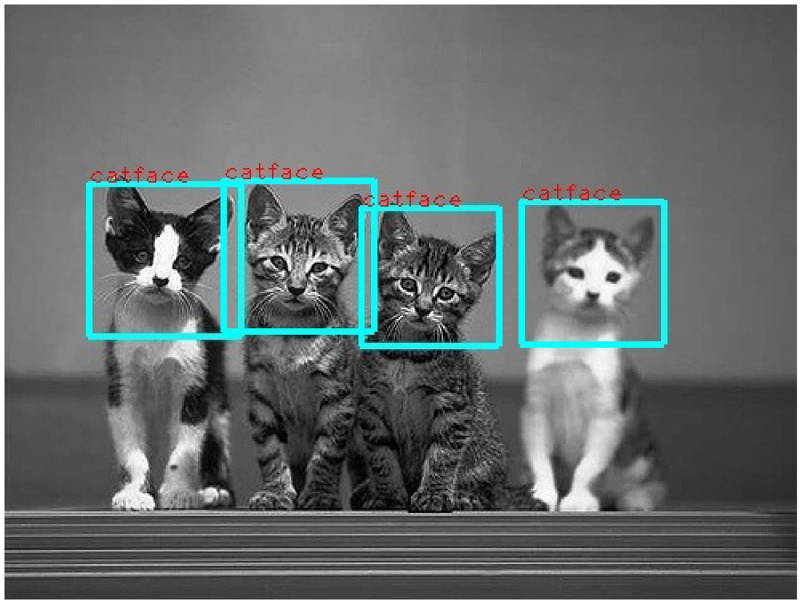

Figure 1:

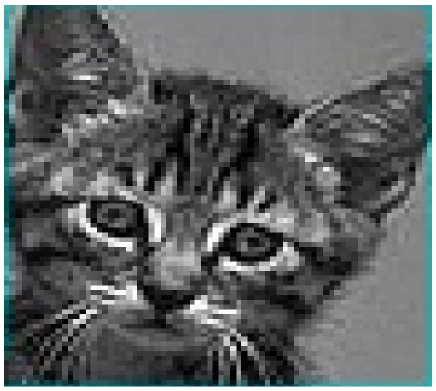

Figure 2:

2.3 SPPnet

R-CNN is slow because it performs a ConvNet forward pass for each object proposal, without sharing computation. Spatial pyramid pooling networks (SPPnets) [2] were proposed to speed up R-CNN by sharing computation. The SPPnet method computes a convolutional feature map for the entire input image and then classifies each object proposal using a feature vector extracted from the shared feature map.

2.4 SPPnet also has notable drawbacks

Like R-CNN, training is a multi-stage pipeline that involves extracting features, fine-tuning a network with log loss, training SVMs, and finally fitting bounding-box regressors. Features are also written to disk.

Accuracy limitation. The fine-tuning algorithm proposed in [2] cannot update the convolutional layers that precede the spatial pyramid pooling. Fixed convolutional layers limits the accuracy of very deep networks.

3 Fast R-CNN

3.1 Advantages

1 Higher detection quality

2 Training is single-stage, using a multi-task loss

3 Training can update all network layers

4 No disk storage is required for feature caching

3.2 Improvement of design

1 Extract features of the entire image at one time

2 Feature extraction, classification, optimization of the object proposal are integrated in a single network

3.3 Main architecture

The following picture [5] shows the main architecture of Fast R-CNN. It takes as input an entire image and a set of object proposals. The network first processes the whole image with several convolutional and max poolinglayers to produce a conv feature map.

Then, for each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map. Each feature vector is fed into a sequence of fully connected(fc) layers that finally branch into two sibling output layers:one that produces softmax probability estimates overK object classes plus a catch-all "background" class andanother layer that outputs four real-valued numbers for eachof theK object classes. Each set of 4 values encodes refinedbounding-box positions for one of the K classes.

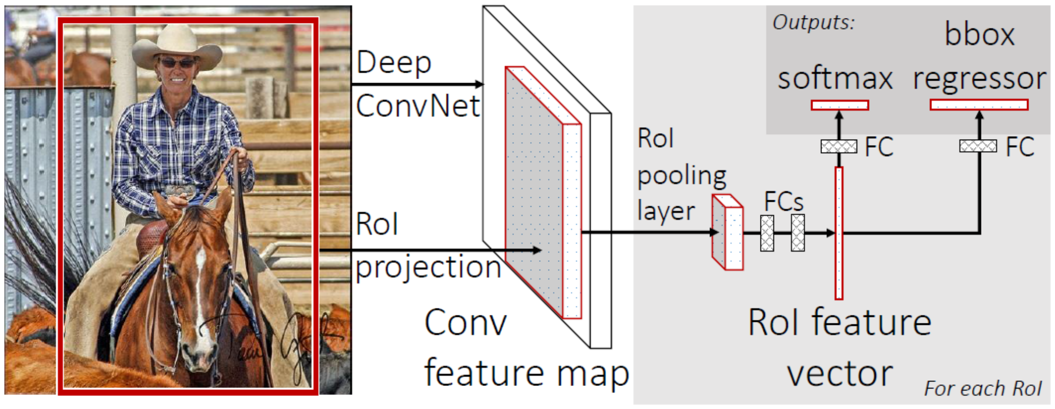

Figure 3:

4 Our Works

Our project is based on the project made by Ross Girshick (github.com/rbgirshick/fast-rcnn, for dataset INRIA). We modified and added some codes, accomplishing to train the neural network and run the object detection program with dataset PKUSVD.

4.1 Brief introduction of directory structure

(directory) caffe-fast-rcnn
containing the framework of Fast-RCNN using caffe implementation

(directory) data
containing the pretrained models and the cache to read files

(directory) experiments
containing the config file and the run log file
some scripts that can get model for ImageNet, and Fast-RCNN model trained by the author
dataset PASCAL-VOC

(directory) lib
some Python interfaces, e.g. “datasets” for loading data from database, “config”for configuration option to train CNN

(directory) matlab
containing matlab interfaces and some python interfaces.
calling matlab interface to implement detection

(directory) models
containing 3 model files for networks of different scales: *CaffeNet* for small

network, VGG16 for large network, VGGCNMM_1024 for middle network

(directory) output
Storing the output file of network training

(directory) proposal
containing two algorithms' implementation for determining object proposals: ACF and Edgebox

containing the Python code for network training and dataset test, very important part of our work

(directory) tools

(directory) toolbox
matlab library for ACF and EdgeBox

(file) trainmergevgg16.sh
a script to run network test

4.2 Implementation

4.2.1 Region Proposals

1 Using Aggregate Channel Features to detect possbile pedestrians, and using Edgebox to detect possible cars.

2 Results are stored in directory /proposal

3 We used this algorithms to get two detectors, cardetector.mat and trainedpersondetector.mat, then we use detectors to process the proposalBounding_box, finishing the region proposal work.

4 Our code is mainly based on getProposalBBox.m.This function, as its name suggests, gets bounding box proposals and saves them in the specific path in a '.mat file'

i.detect car using edgeBox set up some options for edgeBoxes: step size of sliding window search, nms threshold, minScore of boxes to detect and maxBoxes as max number of boxes. We use a trained model 'car_detector' to do this work.

ii.detect person using ACF set up some options for acf feature detector: cascThr as constant cascade threshold, cascCal as cascade calibration, pNms as params for non-mamimal suppresion. Then we use a trained model 'trained_person_detector' to do this work.

iii.postprocess the boxes to the format fast r-cnn needed

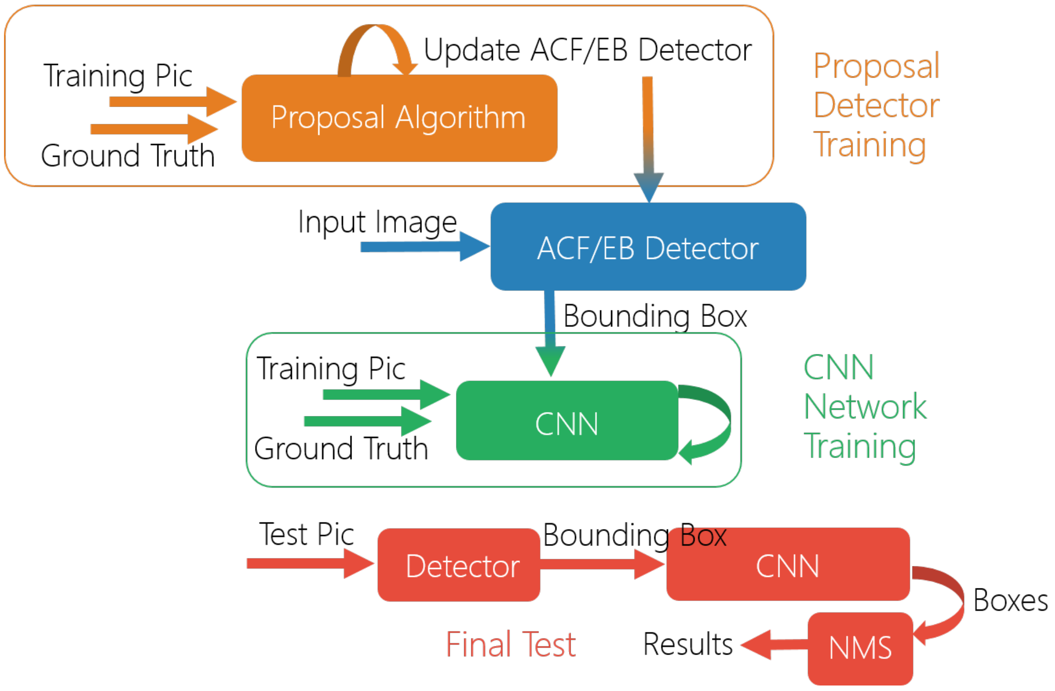
Update ACF/EB Detector
Training Pic
Proposal Algorithm
Ground Truth
Proposal Detector Training
Input Image
ACF/EB Detector
Bounding Box
Training Pic
CNN
Ground Truth
CNN Network Training
Test Pic
Detector
Bounding Box
CNN
Boxes
Final Test
Results
NMS

Figure 4:

4.2.2 Training

1 This is the crucial part of our whole project. We will introduce the function of /tools/train_net.py. It mainly realize following functions: transporting training data, loading nueral network, adjusting the parameter of network, and producing output. This program schedule the entire work of Fine Tuning.

2 Concrete Work

- **i** Receive parameters about network training, deciding which network model we should load
- **ii** Initialize Caffe framwork(including 'gpu','solver','iters','weights','cfg','imdb','rand')
- **iii** Receive the data of object proposals, then generate the imdb format data with function *bisai_multiclass*, and convert it to roidb format with function *get_training_roidb()*.
- **iv** After getting the roidb data and the corresponding neural network, call training func. Training func adjusted the weight of each parameter.The main function is *train_net()*

4.2.3 Data Test

1 Mainly scheduled by /tools/test_net.py

2 Main work:

- **i** Define the test func for a single image:
  - ***** Get object proposals. Obtain data in imdb format
  - ***** Sift the bounding boxes by None-Max-Suppression
  - ***** Set up enviroment for Caffe framework, and score each region in bounding box.
- **ii** Call test func for every image in four groups, produce output.
- **iii** Change format and integrate all output results.

5 Summary

We have succeeded to use Fast-RCNN for object detection in PKUSVD.

We encountered with many difficulties in implementing the project. We gain some experience during the work as well.

Getting familiar with algorithm is the essential part of the work. We also referred to many materials such as the documentation of Fast-RCNN and blogs of other researchers. We also consulted a senior schoolmate, getting much important advice. To face the challenge contacting with so many executive programs including complicated Caffe framework, Python and Matlab interfaces, and to deal with different data format (imdb, riodb), we must make it clear how the files are organized and how the functions relate. Moreover, it also need patience and carefullness when setting up environment, scheduling the order of compilation, linking and calling. Writing "makefile" is troublesome in the project.

6 Result

6.1 Feedback from MLG Lab

Now our Recalls for four test-datas are 0.91/0.87/0.88/0.74 respectively. Average 0.80.

Our Presicions for four test-datas are 0.85/0.70/0.85/0.80 respectively. Average 0.85.

Our fscore for four test-datas are 0.88/0.78/0.87/0.77 respectively.

6.1.1 Detection Time

1 On GPU Cluster of MGL Lab

2 1s per image without warm-up

3 0.1s per image after warm-up

4 $1500 \times 4 \times \; 0.3 = 1800$s

6.1.2 Proposal Time

About 2400s = $1500 \times 4 \times 0.4$

6.1.3 Total

1s/image

6.2 Conclusion from Case Study

1 Some people not annotated by Ground Truth are detected by our Algorithm

2 Mask the picture for promoting the performance

- **i** Mask the area far away from us
- **ii** Mask the area with no possibility to find a person or car
- **iii** Location Info featured?

3 Compared with the example shown by our senior in class(Precision 87.1%,Recall 79.6%), precision rate of our results differs little, but the recall rate is much lower than the example's.

Test result of dataset "southeastern gate" did not meet our expectation. After analysis, we found that there is severe occlusion between people and cars, which is the main cause leading to the low recall rate. Our current program has no good solution in this situation.

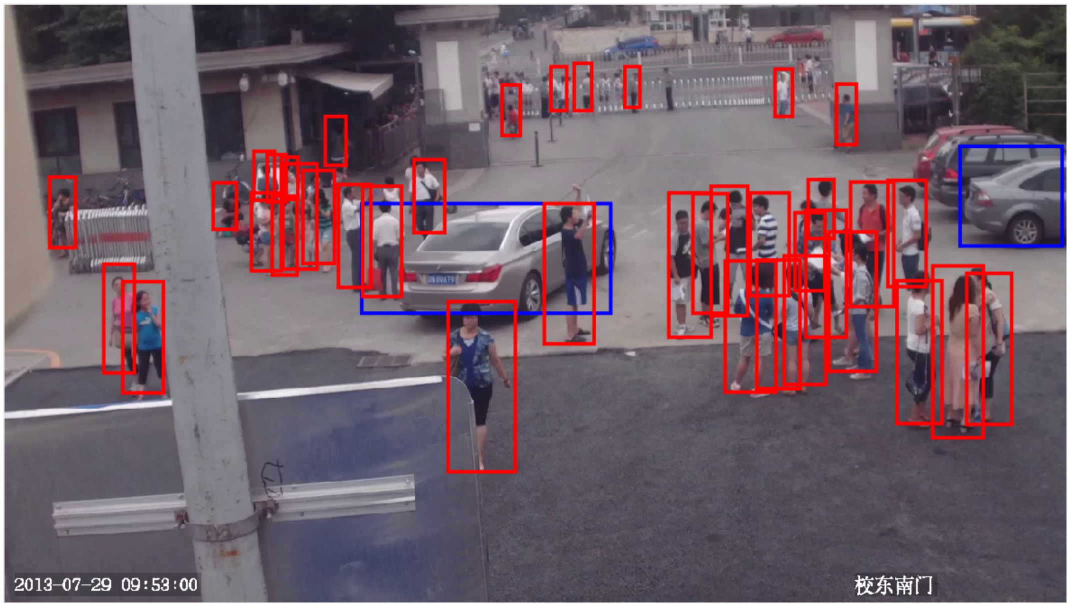

Figure 5: Ground Truth

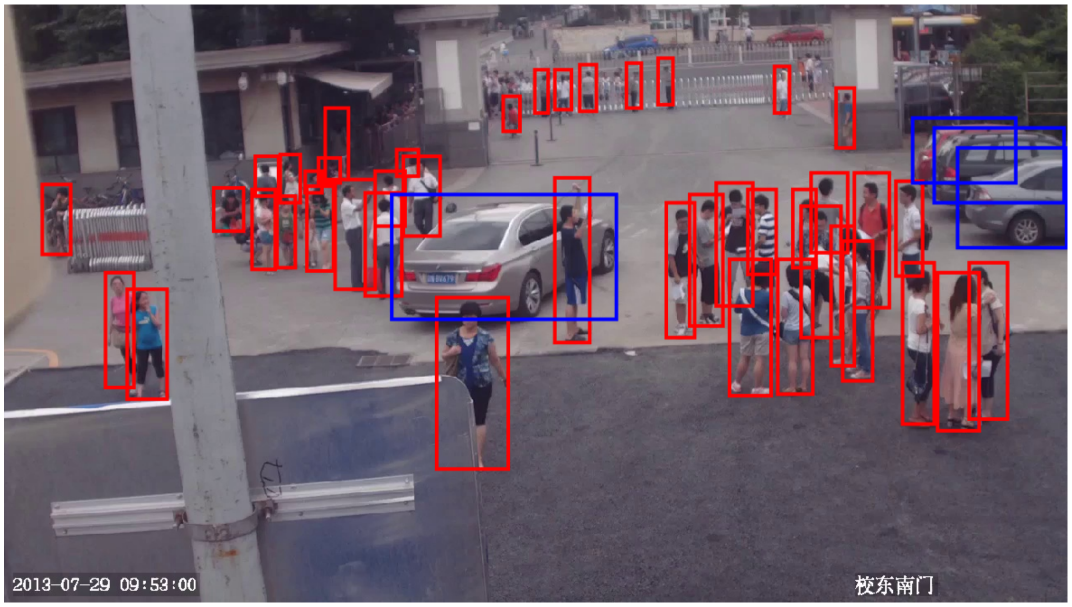

Figure 6: Detection Result

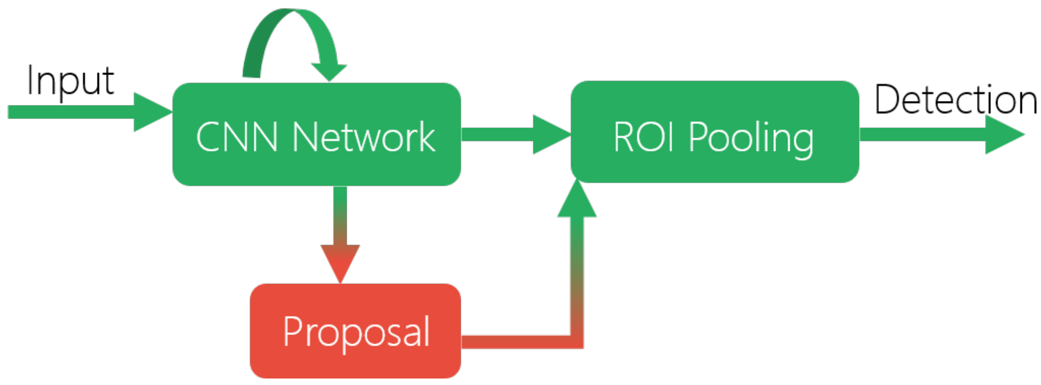

Figure 7:

7 Future Work

We are trying to use Faster-RCNN in training and testing in order to pursue an all the more fast speed.

Faster-RCNN introduced a Region Proposal Network (RPN) that shares full-image convolutional features with the detectionnetwork, thus enabling nearly cost-free region proposals. An RPN is a fully-convolutional network that simultaneously predicts object bounds and objectness scores at each position. RPNs are trained end-to-end to generate high quality region proposals, which are used by Fast R-CNN for detection. With a simple alternating optimization, RPN and Fast R-CNN can be trained to shareconvolutional features.

We have finished data loading, but when we tried to use a fully-connected layer with 3 output units as the last layer, the results are not consistent with our expectation. We are still trying to explore the reason.

8 Reference

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In CVPR, 2014.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In ECCV, 2014.

[3] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding. http://caffe.berkeleyvision.org/, 2013.

[4] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In NIPS, 2012.

[5] R. Girshick. Fast R-CNN. arXiv:1504.08083, 2015.