Object Detection using Tensorflow

SAMIN PAYROSANGARI

SUPERVISER: DIEGO ESTEVES

01/02/2018

Agenda

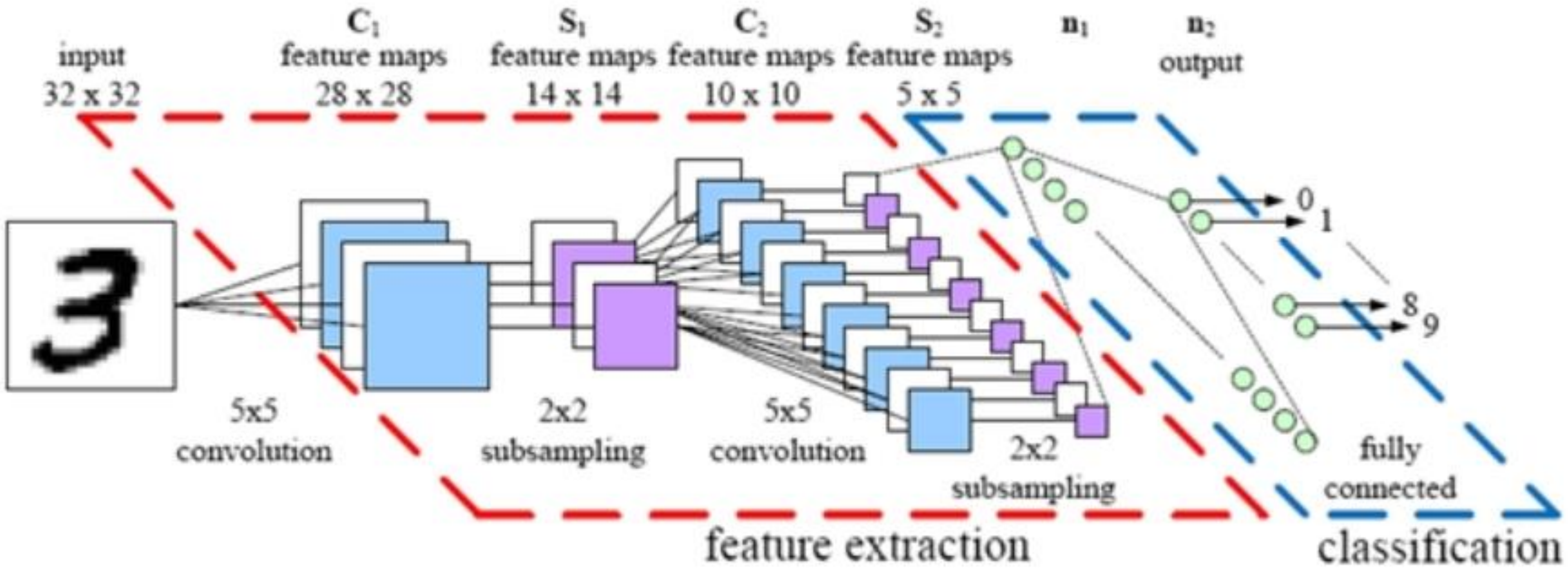
Introduction

Task

Requirments

Results

CNN


input
32 x 32
C1
feature maps
28 x 28
S1
feature maps
14 x 14
C2
feature maps
10 x 10
S2
feature maps
5 x 5
n1
n2
output
0
1
8
9
5x5
convolution
2x2
subsampling
5x5
convolution
2x2
subsampling
fully
connected
feature extraction
classification

CNNs for object detection

Find specific objects in image and draw a bounding box

Extensions of image classification models

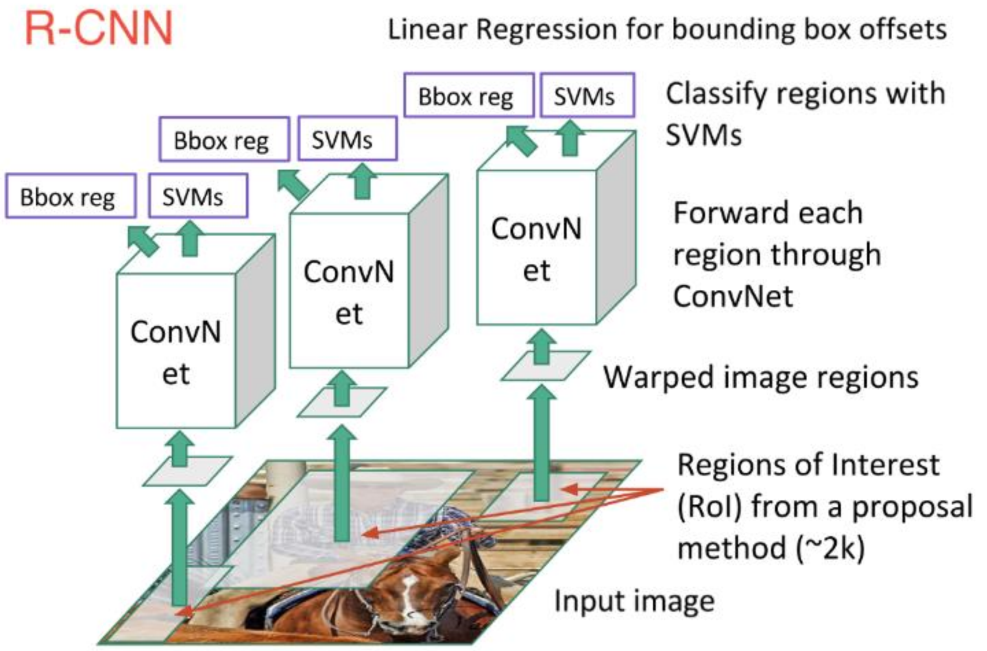
Models:

- **Faster R-CNN (R-CNN → Fast R-CNN → Faster R-CNN)**
- **R-FCN**
- **SSD**

R-CNN

Region-based **C**onvolutional **N**eural **N**etwork:

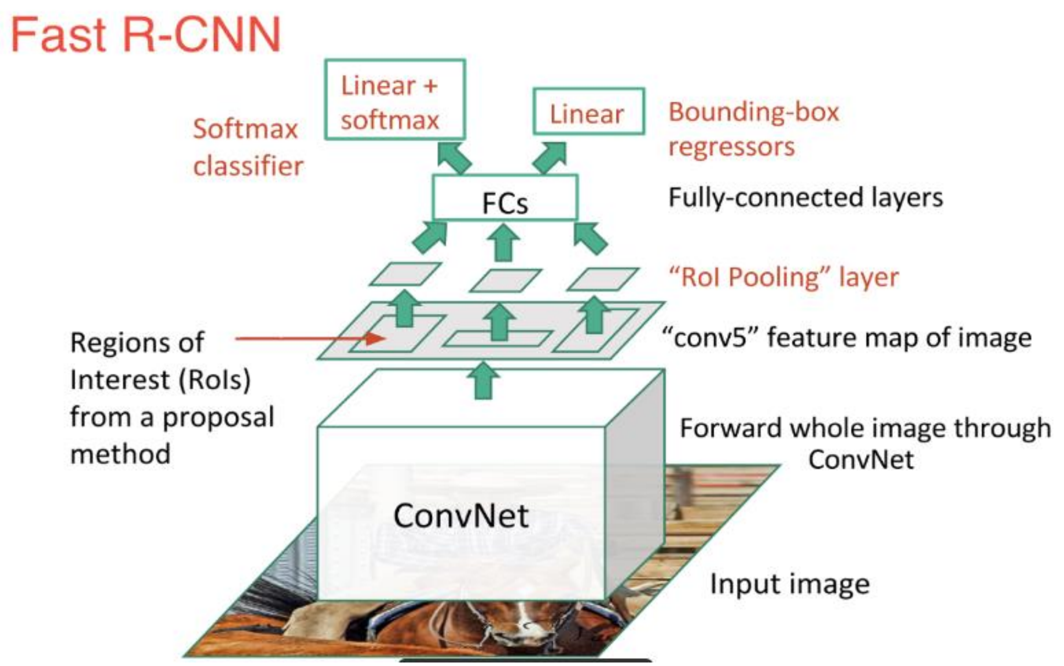
1. Selective search → Scan the input image for possible objects (~2000 **region proposals**)
2. Run **CNN** on top of each region
3. feed each output to:
    a. an SVM to classify the region
    b. a linear regressor to tighten the bounding box



Fast R-CNN

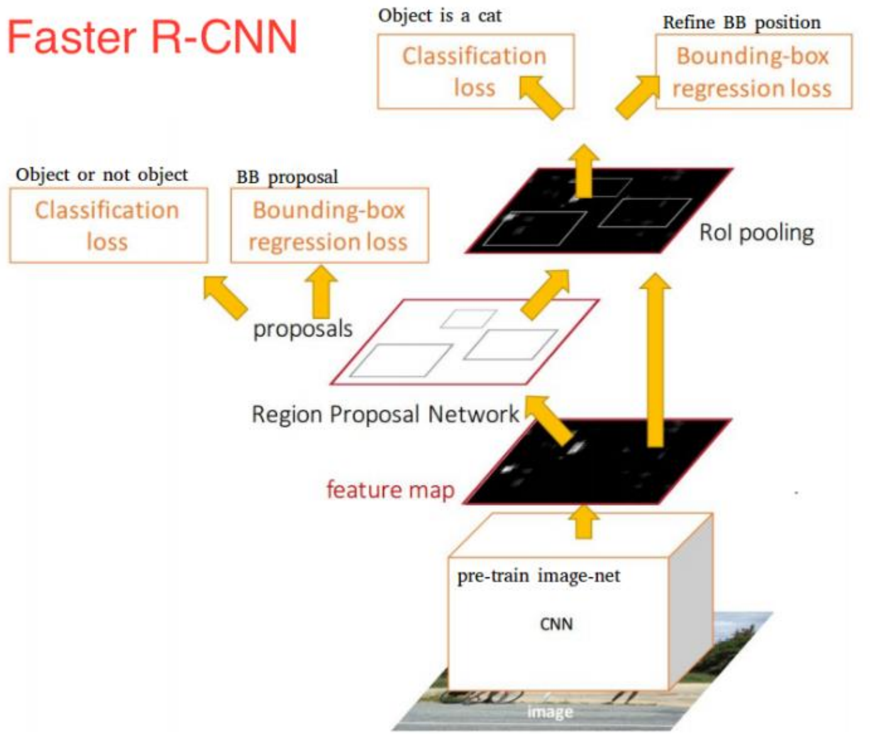
Improved on its detection speed:

- Performing feature extraction over the image **before** proposing regions → running one CNN over the entire image instead
- Replacing the SVM with a softmax layer → extending the neural network for predictions instead of creating a new mode
- Region proposals based on the last feature map of CNN



Faster R-CNN

Main insight → Replace the slow selective search algorithm with a fast neural net: **Region Proposal Network** (RPN)→ **hypothesize object regions and then classify them.**



RFCN (**R**egion-based **F**ully **C**onvolutional **N**et)

Motivation: Increase speed by maximizing shared computation

Fully convolutional

Shares 100% of the computations across every single output → using **position-sensitive score maps** → each score map activates if one specific part of an object is detected.

RFCN simple explanation

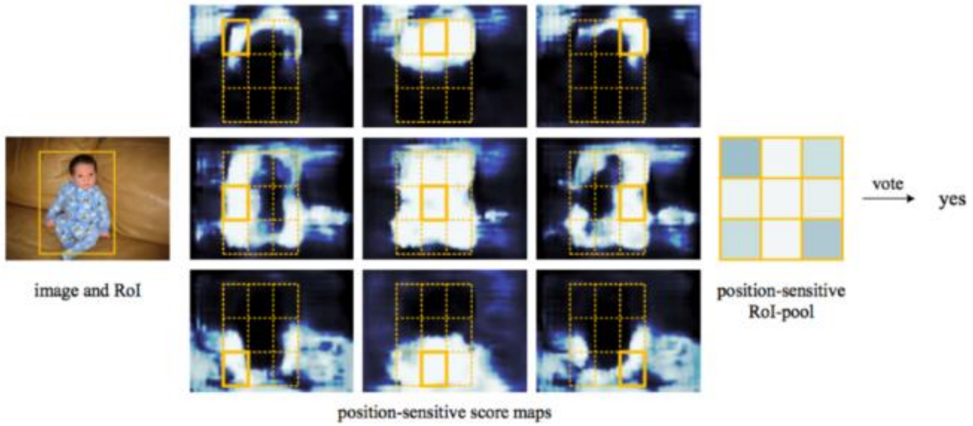
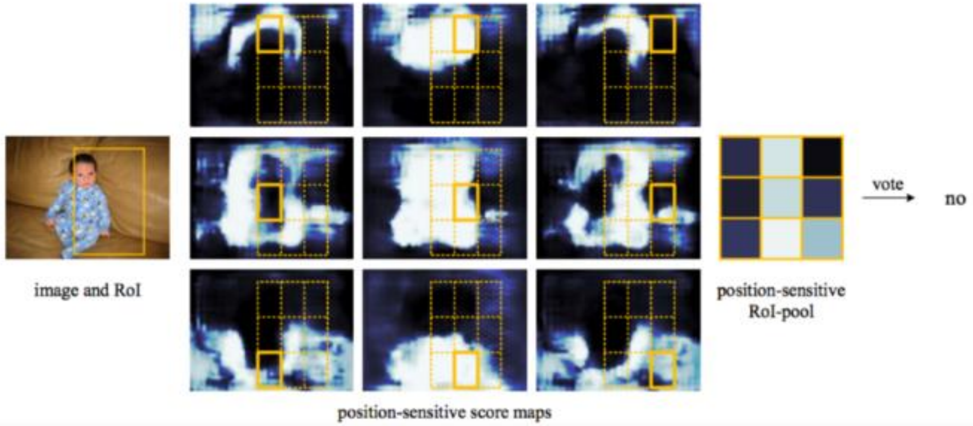


Figure 3: Visualization of R-FCN ($k \times k = 3 \times 3$) for the *person* category.

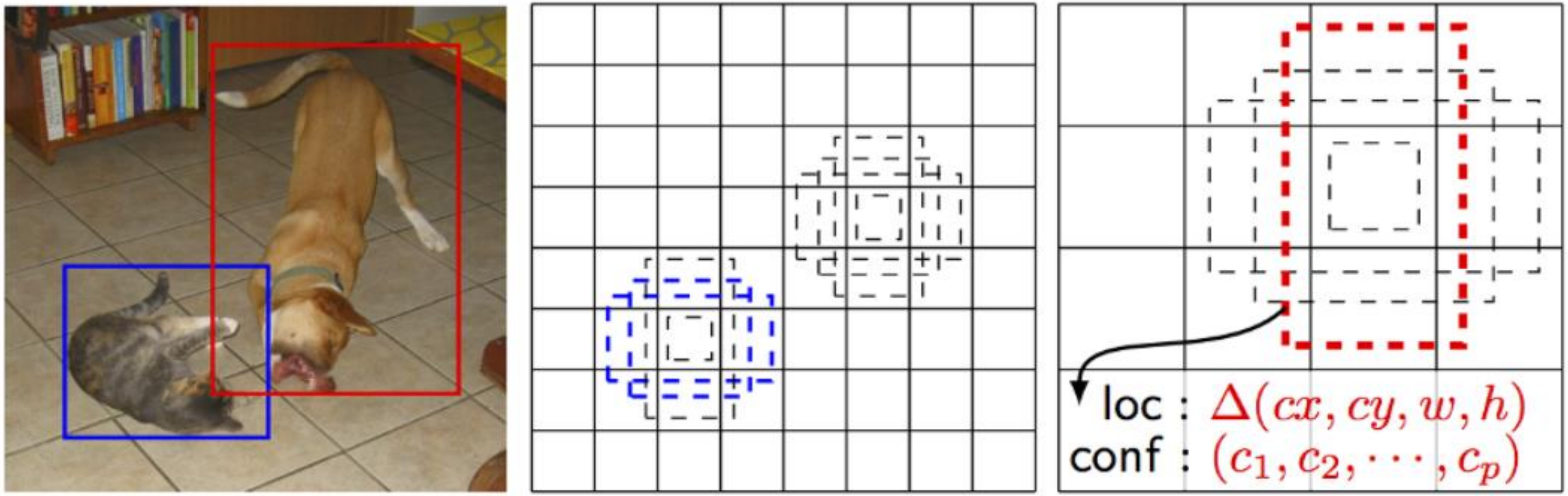


SSD (**S**ingle-**S**hot **D**etector)

Simultaneously predicting the bounding box and the class as it processes the image:

1. Goes through convolutional layers →yielding several sets of feature maps at different scales (e.g. 10x10, then 6x6, then 3x3, etc.)
2. For each location in *each* of these feature maps → considers small set of default bounding boxes.
3. For each box→ **simultaneously** predict the bounding box offset and the class probabilities

SSD simple example



(a) Image with GT boxes (b) 8 × 8 feature map (c) 4 × 4 feature map

Task

Implement object detection models on specific datasets → adidas logo dataset, a dataset of 5 classe obtained from caltech 101 and scene 13 containing person, forest, mountain, highway and building.

Report the results and analysis

Compare the models performance

Solution: Google tensorflow api which comes with object detection models → retrain them with your own dataset.

Requirments

Python 2.7 and 3.6

Tensorflow (python 3.6 compatible)

Object detection API library

Google cloud SDK (python 2.7 compatible)

Google could machine learning engine

Work flow

Turn your dataset into TF records format:

1. Determine the coordinate of bounding boxex around desired objects in training set
2. Turn xml files of training and validation stes into 2 csv format file
3. Turn the csv file into one TF record
4. Generate a label Map for your dataset → Assign labels to int

Workflow (continued)

Prepare google cloud platform project, storage and ML engine

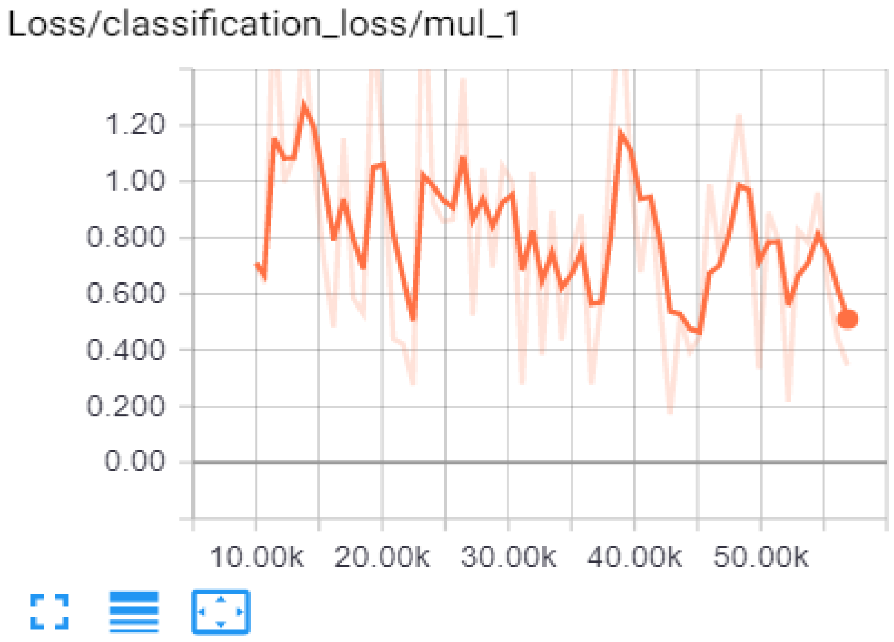
Choose a model → Manipulate the config file of model:

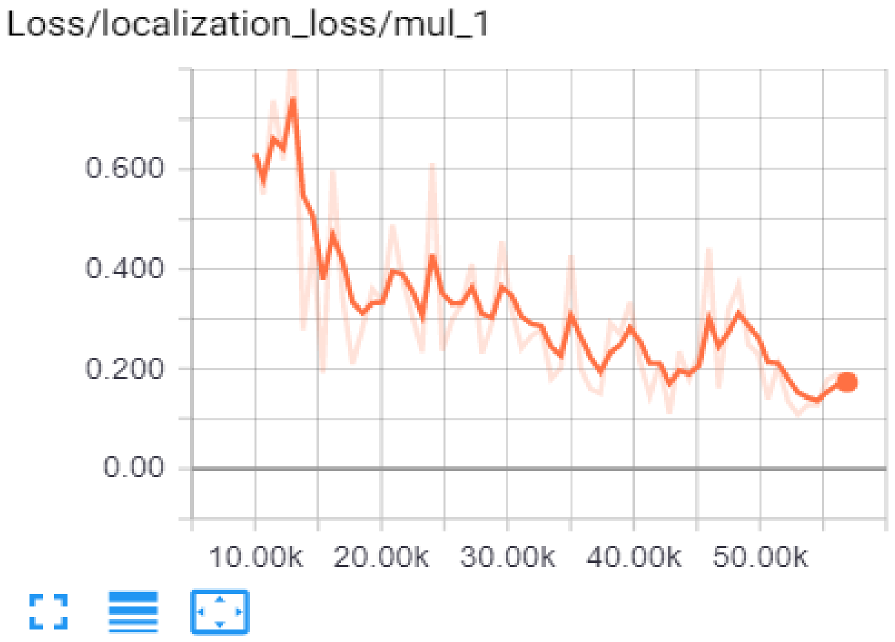
- Config number of classes
- Config optimization and regularization parameters
- Config path to training TF record

Run the training process --> 100000 itarations

Experiment 1: SSD on Adidas logo

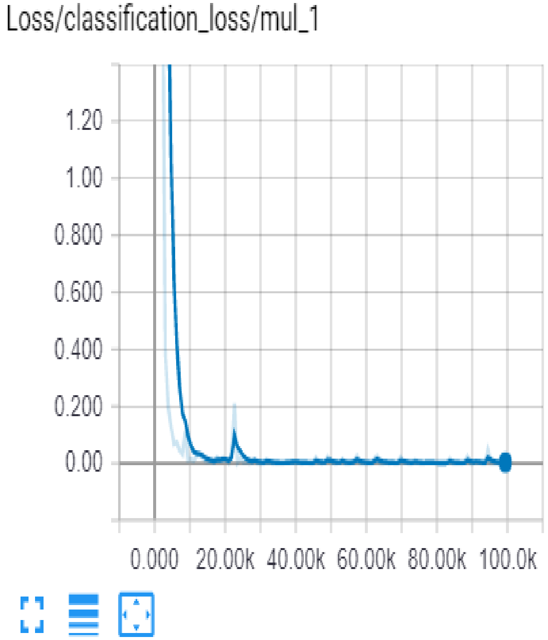
High loss, low performance

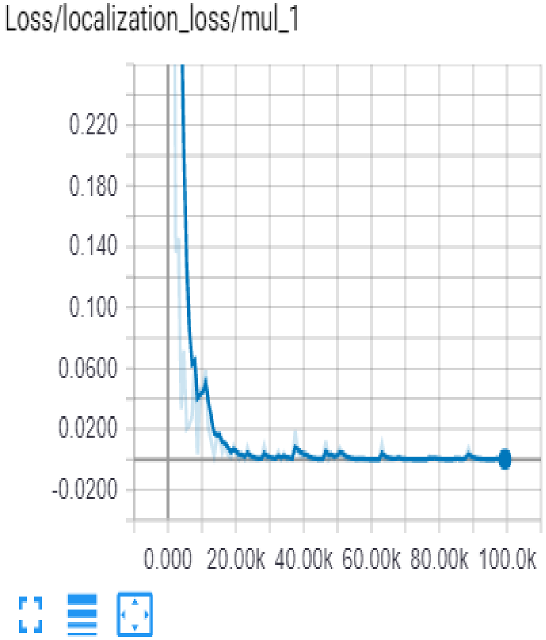




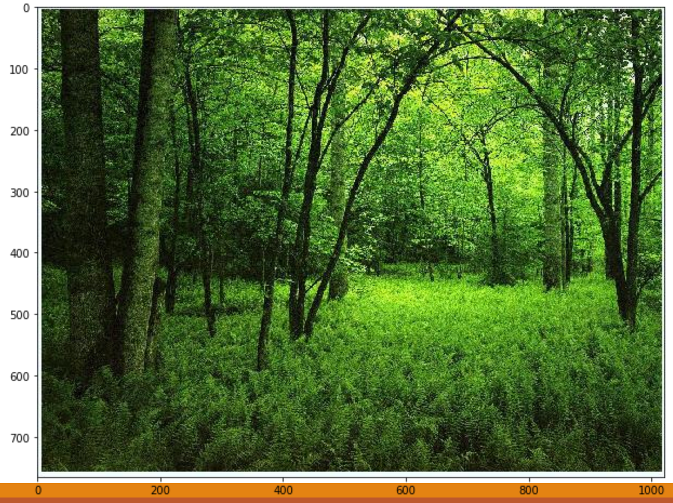
Experiment 2: SSD on 5 classes Data set

1. Adopted to train images for localization
2. Classification performance not so bad
3. Weak in localization



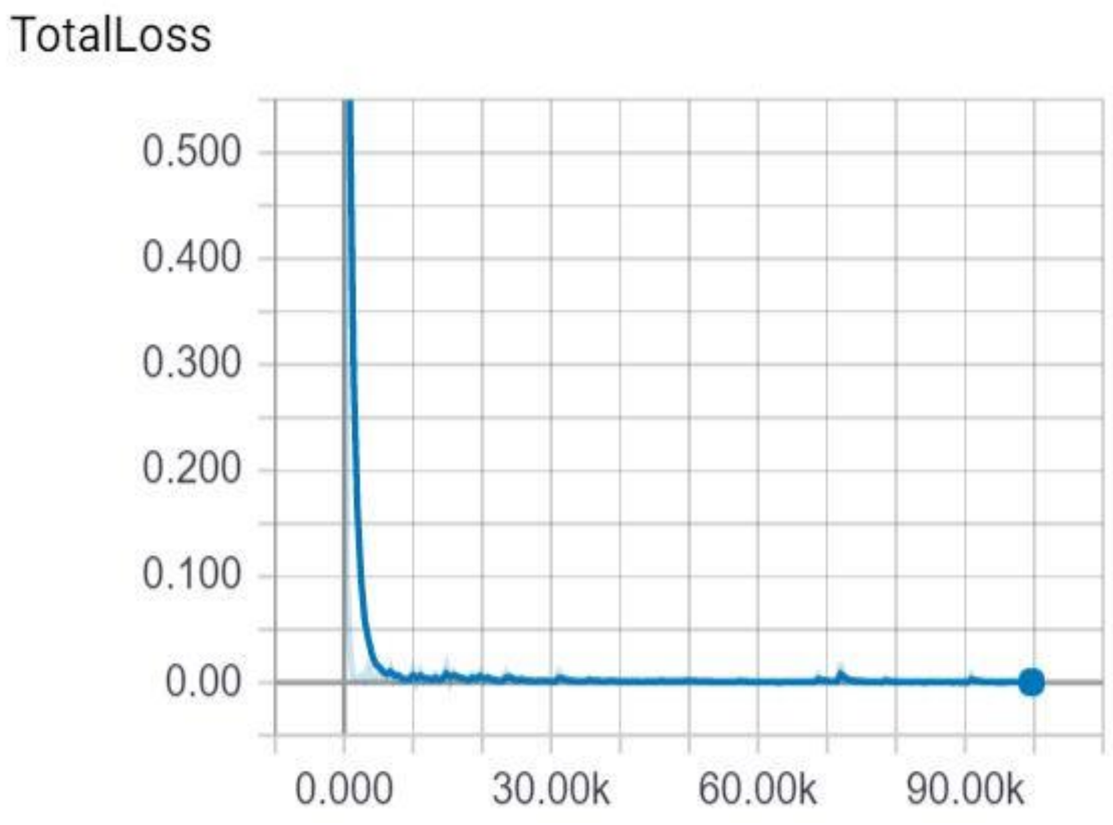


Experiment 2: SSD on 5 classes Data set

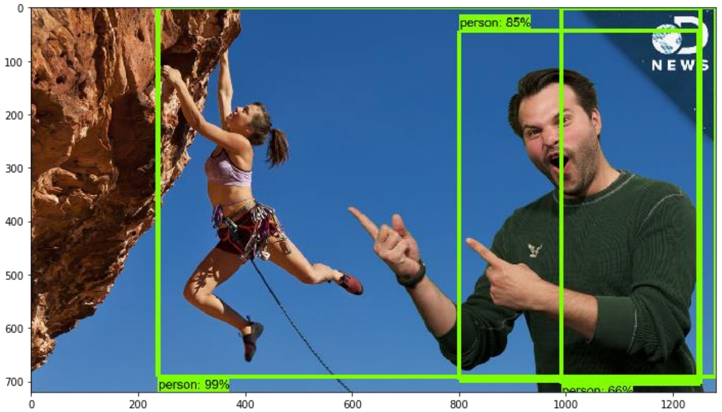


Experiment 3: RFCN on 5 classes dataset

1. Performs better than SSD
2. Better localization
3. Still weak
4. Wrong classification
5. Learned some classes better
6. Has not learned forest class

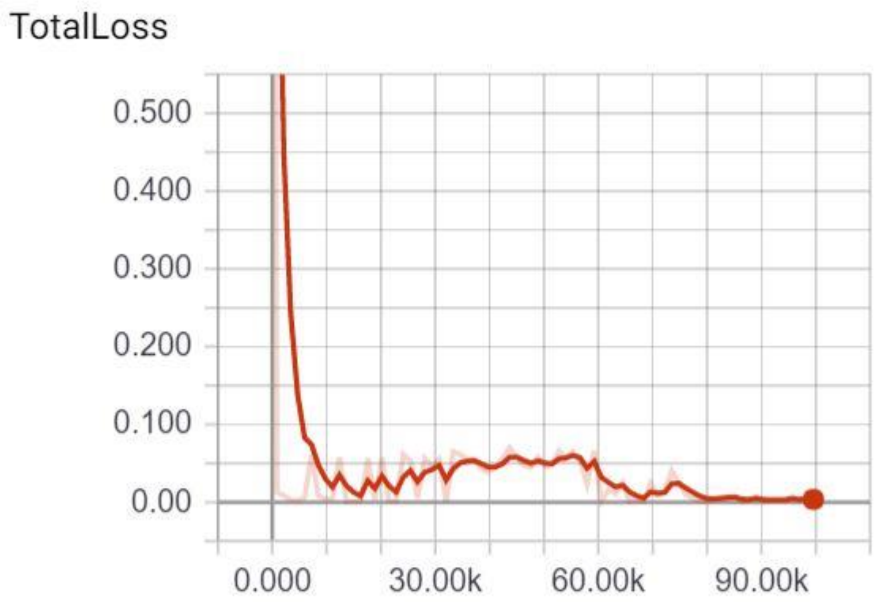


Experiment 3: RFCN on 5 classes dataset

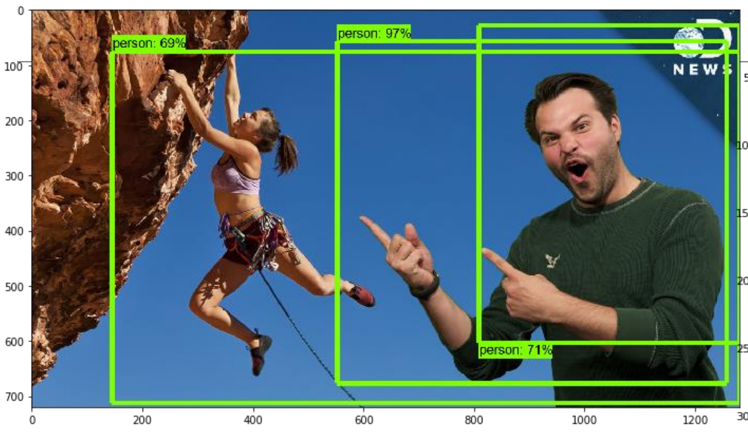


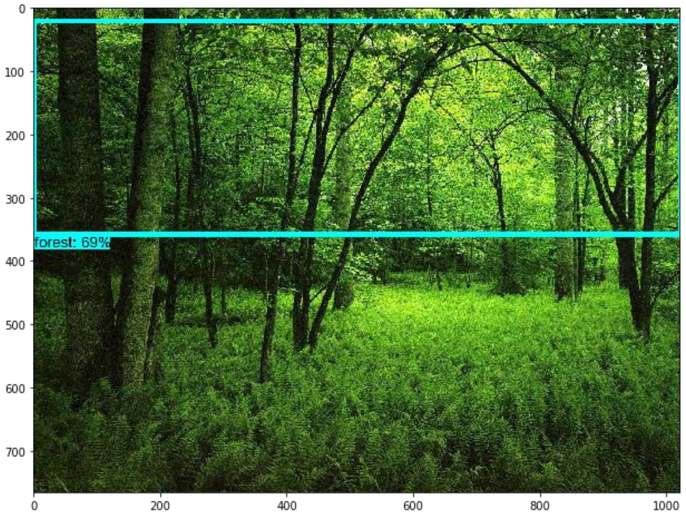
Experiment 4: Faster R-CNN on 5 classes

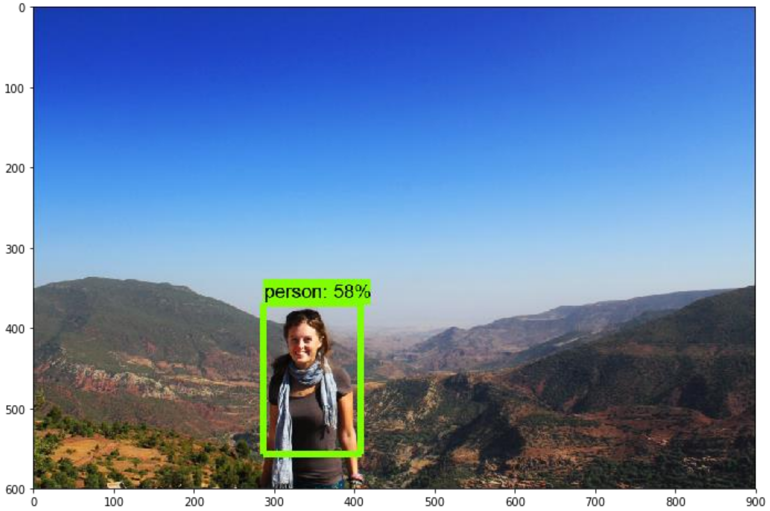
1. Good classification
2. Good localization
3. Still some wrong classification → Could improve by increasing the number of iterations



Experiment 4: Faster R-CNN on 5 classes







Conclusion

Ranking by performance:

1. Faster R-CNN
2. RFCN
3. SSD

Increasing the number of iterations for SSD does not seem to enhance the performance.

Increasing the number of iterations for RFCN might enhance the performance.

Faster R-CNN is slower but more powerful.

References

[1] https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9

[2] https://medium.com/google-cloud/object-detection-tensorflow-and-google-cloud-platform-72e0a3f3bdd6

[3] https://cloud.google.com/solutions/creating-object-detection-application-tensorflow

[4] https://towardsdatascience.com/how-to-train-your-own-object-detector-with-tensorflows-object-detector-api-bec72ecfe1d9

[5] https://github.com/tensorflow/models/issues/2739