Wrocław University of Science and Technology
Faculty of Electronics, Photonics and Microsystems

Object detection with deep learning methods
Actually transfer learning implementation for custom object detection

Author: Maroine Trabelsi

Date: January 18, 2022

Class: Intermediate Project

Supervisor: Ph.D. Witold Paluszyński

Abstract

The proposed project is set with the goal of achieving a functional object detection application with the usage of deep learning methods basing on the MS Coco dataset [4].
Moreover it is presented an approach for custom object detection, which relies on transfer learning from the coco dataset pretrained model. The custom dataset chosen for this purpose is a collection of road signs [9] of four different classes. The proposed approach has been tested for different CNN architecture parameters and the generated results are described and analyzed in the results section.

1 Introduction

Object detection is a computer vision technique that allows to identify and locate certain objects in an image or a video. Object detection techniques are widely adopted in many fields, some of which are surveillance and security, traffic monitoring, video communication and robot vision. Object detection can be broken down into machine learning-based approaches and deep-learning based approaches. The project report will be focused on the second category approach, deep learning which employs convolutional neural networks (CNN).

2 Assumptions

2.1 Object detection

Object detection, regardless of whether performed via deep learning or other computer vision techniques, has three primary goals. Given as input an image or frame (for videos) it is expected to obtain:

1. A set of cordinates $(x_{min}, y_{min}, x_{max}, y_{max})$ forming a bounding box around the interested object
2. A class label associated with the bounding box
3. The confidence score associated with the bounding box and class label.

An example of the expected output is presented below:

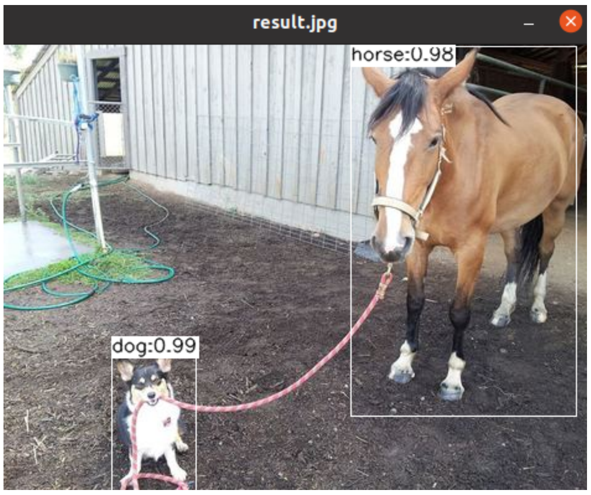


Figure 1: An output image of two detected objects (animals) with the drowned bounding boxes, associated labels (”dog”,”horse”) and the respective detection accuracy $(0.95, 0.98)$

2.2 Deep learning detection methods

The set of methods used for object detection is categorized in two main types:

- Two-stage
  The process relies on two tasks:
  1. Find the number of objects on the scene
  2. Classify every single object by estimating its shape with a bounding box

  In such approach the approximate object regions are proposed using deep features before these features are used for the classification. The first category achieves the highest detection accuracy but is typically slower than one-stage type. Algorithms identified in such subset are RCNN, Fast RCNN and Faster RCNN among others.

- One-stage
  The second category methods predict the bounding boxes over the images without the regional proposal step. Such process consumes less time at the cost of accuracy, which is a good solution in case of real-time applications. Some popular algorithms of that set are YOLO (You Only Look Once) and SSD (Single Shot Detector) among others.

  In this project the one-stage detectors will be taken into consideration for the transfer learning task, which are structurally simpler.

2.3 Transfer learning

In an ordinary approach training an object detection model may take long times, however the transfer learning technique allows to take a model which has been already trained (pretrained) and be used as a starting point to create a new model. In the following project this technique will be applied to the already pretrained models (on COCO dataset) in order to re-use the base learning process for the custom objects detection.

3 Constraints

Object detection comes with multiple challenges.

1. The first major complication of object detection is the **speed for real time detection**. The applied algorithms need to not only accurately classify and localize important objects, they also need incredibly fast prediction time in order to meet the real-time demands of video processing.

2. When training a custom object detector the **accuracy of the predicted bounding boxes- Intersection of union (IoU)** plays a crucial role. The convolution neural network, when trained, generates thousands of anchor boxes for each predictor that represents the ideal location of the object(s). For each predicted anchor box the intersection over union metrics is applied, which consist on determining the area of union between the real bounding box (also called ground truth) and the predicted anchor box by the equation:

$$IoU = \frac{Areaofoverlap}{AreaofUnion} \tag{1}$$

   If the IoU value is low, it means that the predicted anchor box does not fit the real object and the estimation is poor. In case of a high IoU value (+ 90%) then the estimation is good or excellent. An example has been presented below from [10]:

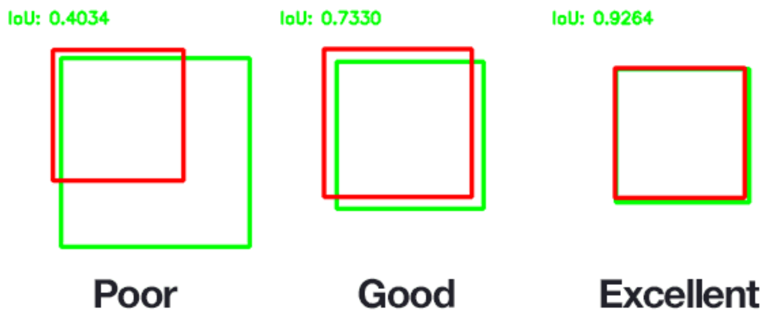


Figure 2: An example of computing Intersection over Unions for various bounding boxes
An evaluation of the Intersection over Union for the custom object detection has been presented further in the document.

3. **Dataset labelling**
   A limited amount of annotated data available for object detection can be a big issue when the goal is to obtain a good estimator. Object detection datasets typically contain ground truth examples. When such data is not available gathering ground truth labels can be a tedious work. For the purpose of the project a dataset of traffic signs has been chosen from Kaggle [9] with already labeled images (.xml files).

4. **Training - computational requirements** Training models is a hardware intensive task, and in order to obtain good results in a short time a decent GPU is required. Having the possibility of running the training step using a GPU will make sure that the computation of the neural network goes smoothly. For this reason the Google Colab platform has been utilized which offers, among others, GPU like Nvidia Tesla P100, T4 and K80 with 16GB,16GB and 12GB RAM respectively.

4 Summary

The project idea has been developed and extended through the whole mainly in three milestones:

1. Firstly the topics related to computer vision, in particular OpenCV [5] and basic neural networks applications have been gathered and analyzed. A first version of object detection for images basing on YOLO has been developed for the first milestone.
2. In the next months various methods, models and dataset have been analyzed and applied for practical experiments (such as Tensorflow, Caffe and DarkNet models). Such experiments included also the processing of video frames, where it was necessary to optimize the time delay between each video frame in order to obtain sufficient estimation results as reduced computational time. An object detection application for images and videos basing on pre-trained models, at that time was, already developed.
3. Finally, after a review of state of the art solutions for object detection, a few papers [1],[2] have been taken into consideration for an additional project expansion - transfer learning implementation for custom object detection. This final expansion gave the ability to train a custom dataset and most important evaluate different models and neural network architecture parameters.

5 Implementation

In the following section the custom object detection proposed approach will be presented in N sections

5.1 Preparation of the dataset

The dataset chosen for this task is the **Road Signs Object Detection** dataset [9] (example images represented below) and it contains:

- 877 .png images of 4 classes:
  1. traffic light
  2. stop sign
  3. speedlimit sign
  4. crosswalk sign
- 877 .xml annotations containing:
  1. bounding box coordinates
  2. class name

Figure 3: Sample images from the Road Signs object detection dataset

Next the dataset has been divided into two sections:

- **Training set**
  The set used for training the model containing 702 images and corresponding labels (80%)
- **Test set**
  The set used for testing the model containing 175 images and corresponding labels (20%)

5.2 Preparation of Tensorflow models files

Tensorflow models, which are used to train a model, require the .csv file format of the labels and for this cause the .xml labels files have been exported to the .csv format. Additionally a separate source of record for class annotations is required and for this the label map is generated. The label map is an annotation of the number of class with its respective label name. For this case, the file looks like:

```
item {
    id:1
    name: 'trafficlight'
}
item {
    id:2
    name: 'stop'
}
item {
    id:3
    name: 'speedlimit'
}
item {
    id:4
    name: 'crosswalk'
}
```

Finally having the files exported, in order to meet the requirements of Tensorflow, the TFRecords have been generated. The TFRecord file stores the data of images and annotations as a sequence of binary strings readable by Tensorflow models and it has been generated one for each section: training TFRecord and testing TFRecord.

5.3 Configuration of the pre-trained model

In order to apply the transfer learning technique it is required to have a pre-trained model, of which the learning process is deducted from the neural network layers skipping the last layers of the network. The pretrained model used for the task is

- Single Shot Detector applied to Mobilenet CNN version 1.0 for COCO dataset (ssd_mobilenet_v1_coco)

The evaluation metrics published by Tensorflow regarding to the model proposed have been presented below:

Model name	Speed (ms)	COCO mAP	Outputs
ssd_mobilenet_v1_coco	30	21	boxes

Where **Speed** reports the model speed running time in ms for a 600x600 image and **COCO mAP** is the detector performance on the subset of the COCO validation set.

5.4 Parametrization of the neural network

In the next step different configurations of the underlying neural network have been applied, mostly the changes are related to the batch size (number of training examples utilized in one iteration) and the number of training steps. The chosen parameters and the corresponding values have been represented below:

Model name	Number of steps	Batch size
ssd_mobilenet_v1_coco	50.000	8
ssd_mobilenet_v1_coco	50.000	16

Once the network configuration has been modified accordingly, the training process has started. The accuracy metrics obtained from each model on the dataset have been added in the next sections.

5.5 Object detection algorithm implementation

The final step of the project recalls the first one deployed, which is the algorithm for object detection. The main steps of the algorithm are represented below:

1. Load the classes names (from COCO dataset or from the Traffic Signs dataset)
2. Load the model configuration and the corresponding weights (available pre trained models or generated from custom)
3. Read the input image and convert it into a blob data in order to feed the blob variable into the network.
4. Evaluate the output layer names
5. Remove the bounding boxes with low confidence (lower than a provided threshold value)
6. Draw prediction bounding boxes and the corresponding label name.

6 Results

6.1 Training results

As described in section 5.4 different batch sizes have been chosen in order to evaluate the trained models. Those values will be recapped just below updated with the results obtained. The training process has been performed on a Google Colab instance machine having provided **GPU Tesla P100 16GB RAM**. Below the results have been published in form of a table and graphs for intuition.

n	Number of steps	Batch size	$t_{training}[h:mm]$	$Loss_{finalstep}$	mAP
1	50.000	8	1:51	1.5790558	47.54
2	50.000	16	3:14	1.3244216	49.56

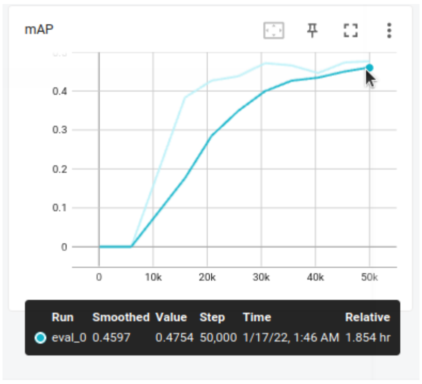
6.2 Object detection results

In the final step the generated models have been implemented in the object detection algorithm and tested for an efficiency evaluation. For the purpose of find the differences between the two models (if any) two different images have been taken and processed by the object detection algorithm. The output images results have been added below.

In the first model output image for traffic light the detected objects are three on five with a probability of $0.78, 0.68, 0.69$, while for the second model output the traffic lights detected are five on five with probabilities $0.47, 0.86, 0.71, 0.51, 0.46$. For the second image the signs are detected for both the models. However in case of the first model the speed sign was detected as a traffic light (0.52) and the cross walk with 0.95 accuracy, but with an additional false positive (traffic light) in the air. While for the second model the speed limit sign is detected with an accuracy of 0.99 and the crosswalk 0.48.

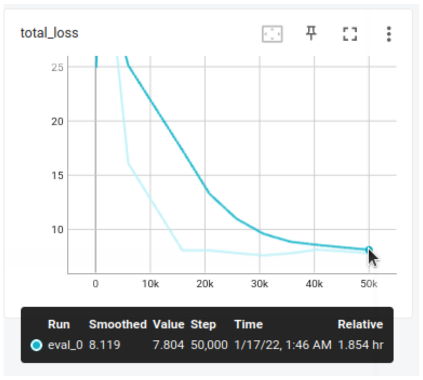
7 Conclusion

Both the models were trained with the same amount of images (same ratio for training and testing) and samples. The batch size change of the neural network did impact on the mAP value of the model as well on the final loss value. It has to be said that in case of a twice bigger batch size the training time as well increases almost twice. The obtained results confirm, in part, that a model trained on a higher number of training examples in one iteration (batch size) give usually better results when it comes to localization. Further analysis could be done trying to increase the number of steps and for other methods.



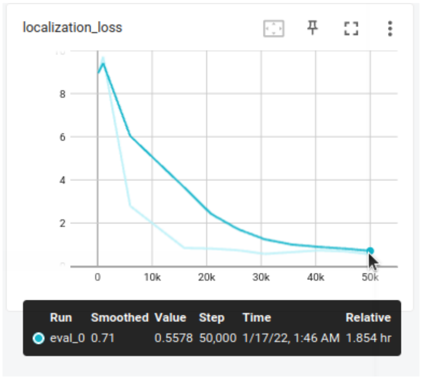
Run	Smoothed	Value	Step	Time	Relative
eval_0	0.4597	0.4754	50,000	1/17/22, 1:46 AM	1.854 hr

(a) mAP in accordance to sample value iteration



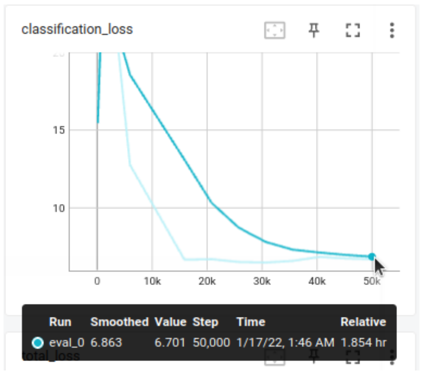
Run	Smoothed	Value	Step	Time	Relative
eval_0	8.119	7.804	50,000	1/17/22, 1:46 AM	1.854 hr

(b) total loss in accordance to sample value iteration



Run	Smoothed	Value	Step	Time	Relative
eval_0	0.71	0.5578	50,000	1/17/22, 1:46 AM	1.854 hr

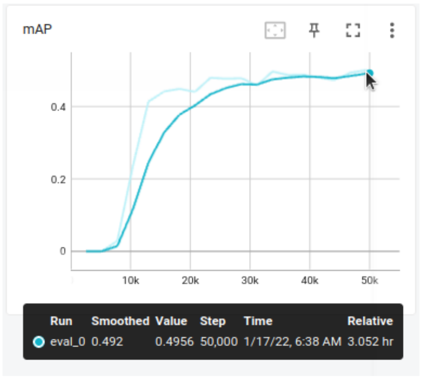
(c) localization loss in accordance to sample value iteration



Run	Smoothed	Value	Step	Time	Relative
eval_0	6.863	6.701	50,000	1/17/22, 1:46 AM	1.854 hr

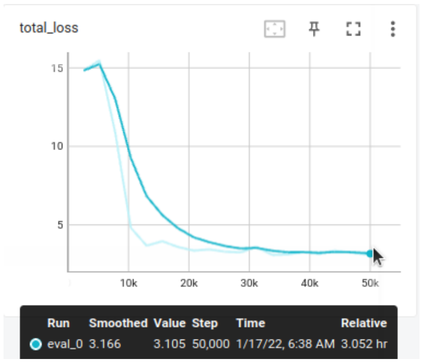
(d) classification loss in accordance to sample value iteration

Figure 4: Tensorboard graph metrics values for the first network configuration (batch size 8)



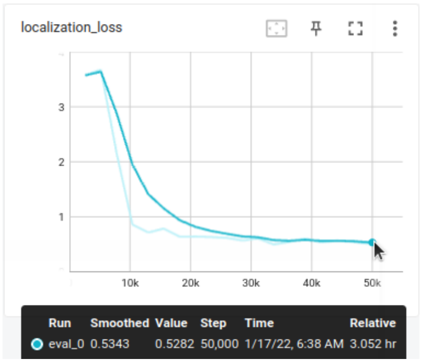
Run Smoothed Value Step Time Relative
eval_0 0.492 0.4956 50,000 1/17/22, 6:38 AM 3.052 hr

(a) mAP in accordance to sample value iteration



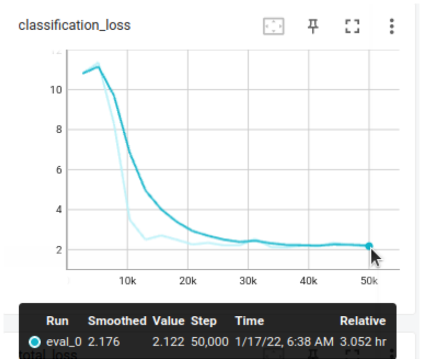
Run Smoothed Value Step Time Relative
eval_0 3.166 3.105 50,000 1/17/22, 6:38 AM 3.052 hr

(b) total loss in accordance to sample value iteration



Run Smoothed Value Step Time Relative
eval_0 0.5343 0.5282 50,000 1/17/22, 6:38 AM 3.052 hr

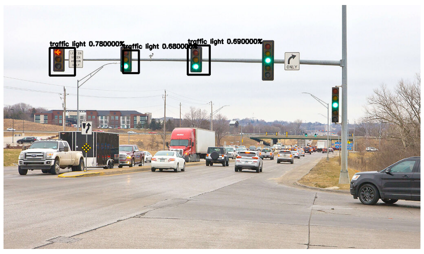
(c) localization loss in accordance to sample value iteration



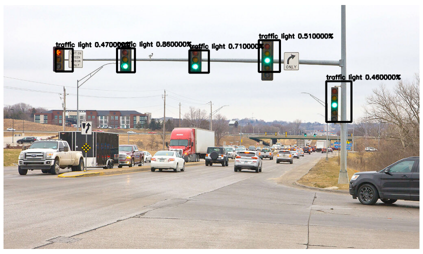
Run Smoothed Value Step Time Relative
eval_0 2.176 2.122 50,000 1/17/22, 6:38 AM 3.052 hr

(d) classification loss in accordance to sample value iteration

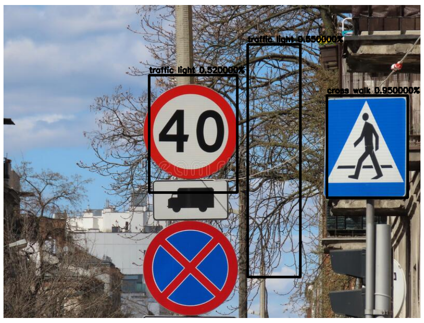
Figure 5: Tensorboard graph metrics values for the second network configuration (batch size 16)



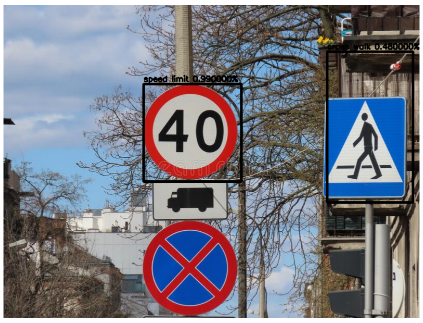
(a) traffic lights detected by first model iteration



(b) traffic lights detected by second model



(c) traffic signs detected by first model iteration



(d) traffic signs detected by first model

Figure 6: Output images from object detection algorithm for two trained models: batch size 8 on the left, 16 batch size on the right

8 Materials and resources

References

[1] Alvaro Arcos-Garc ıa, Juan A. Alvarez-Garc ıa, Luis M. Soria-Morillo, Eval- uation of Deep Neural Networks for traffic sign detection systems, Neurocomputing (2018), doi: https://doi.org/10.1016/j.neucom.2018.08.009 https://doi.org/10.1016/j.neucom.2018.08.009

[2] Real-time Detection and Recognition of Traffic Signs in Bangladesh using YOLOv3 Detector

[3] Building Computer Vision Projects with OpenCV 4 and C++: Implement complex computer vision algorithms and explore deep learning and face detection https://books.google.pl/books?id=naOPDwAAQBAJ

[4] MS COCO dataset https://cocodataset.org/overview

[5] Open CV documentation https://docs.opencv.org/4.5.4/

[6] Open CV C++ Tutorial https://www.opencv-srf.com/p/introduction.html

[7] A Gentle Introduction to Object Recognition With Deep Learning, Author: Jason Browniee https://machinelearningmastery.com/object-recognition-with-deep-learning/

[8] Deep Learning with OpenCV DNN Module: A Definitive Guide, Author: Satya Mallick https://learnopencv.com/deep-learning-with-opencvs-dnn-module-a-definitive-guide

[9] Road Sign Dataset https://makeml.app/datasets/road-signs

[10] PyImageSearch https://www.pyimagesearch.com/