“detect … or detect NOT!”

https://www.pinterest.ca/pin/5207355798940572/

ELEC 475

Lab 4 – YODA: You Only Detect Anchors

Lab 4 – YODA

Michael Greenspan

Contents

1. Introduction

In this lab you will implement an object detection method, to detect instances of the "Car" class in the Kitti8 dataset.

2. Task

Your task is as follows:

2.1 Step One – Visualize Kitti

Download and unzip the Kitti8 dataset from the G drive.

The dataset contains 7148 image and label files, partitioned into a training set (6000 image and label files) and a test set (1148 image and label files). The images are of street scenes, acquired from a dashcam. The label files contain the bounding boxes and labels of the objects in the scene. In total, there are 9 different types of object labels: 'DontCare': 0, 'Misc': 1, 'Car': 2, 'Truck': 3, 'Van': 4, 'Tram': 5, 'Cyclist': 6, 'Pedestrian': 7, 'Person_sitting': 8.

The dataset and bounding boxes can be visualized using the **`showKitti.py`** module, using the syntax in the **`showKitti.txt`** script.

2.2 Step Two – Generate Yoda Dataset

Yoda is concerned only with detecting the 'Car' class. To accomplish this, its convenient to generate a new dataset, comprising regions of interest (ROIs) from Kitti. The new YODA dataset will contain ROIs sampled from Kitti, with each ROI labelled as either a 'Car' (1), or 'NoCar' (0) class.

The provided module **`KittiToYodaROIs.py`** can be used to generate the YODA dataset, using the script **`KittiROIs.txt`**. This module subdivided the Kitti images using a regular grid, as specified in the **`KittiAnchors.py`** module. Each ROI is checked against the Kitti image labels, measuring the IoU of the ROI with a Kitti 'Car' object. If this IoU value exceeds a threshold, then the ROI is labelled as a 'Car' – otherwise, it is labelled as a 'NoCar'.

Generate both train and test YODA partitions, from the train and test Kitti partitions respectively.

Hint: The default anchor values specified in the **`KittiAnchors.py`** module worked well for me. You can change them if you wish.

2.3 Step Three – Train and Test a YODA Classifier

Once you've generated the YODA dataset, train a classifier. This classifier can be based on the model you developed for Lab 3, or you can use an existing model from the PyTorch collection. Save your loss plot, using the test partition for validation.

Once trained on the YODA train partition, test it on the YODA test partition.

Hint: I used the default PyTorch ResNet-18 model. It converged in ~40 iterations.

2.4 Step Four – Train and Test YODA

Build a YODA model that does the following:

1. Subdivide a Kitti image into a set of ROIs. Save the bounding box coordinates for each ROI.
2. Build a batch from the ROIs, and pass the batch through the YODA Classifier that you trained in Step 3.
3. For each ROI classied as a 'Car', calculate its IoU score against the original Kitti image.

Test this on a few images, displaying the results (i.e. rendering the detected 'Car' bounding boxes) on the original Kitti image.

Once its working, calculate the mean IoU value for all detected 'Car' ROIs, over the complete Kitti test partition.

Hint: Much of the code that was used in Step 2, can be repurposed here.

Hint: All ROIs from a single Kitti image can be composed into a single batch, and sent to the model in one call.

3. Deliverables

The deliverable as described below comprises the following:

- The completed code;
- A report.

3.1 Completed Code

In addition to the code itself, provide the following four scripts to train and test your code (e.g. from the PyCharm terminal):

`step3_train.txt`

`step3_test.txt`

`step4_train.txt`

`step4_test.txt`

3.2 Report

You report should be relatively brief (1-2 pages), and include the following information:

- Step 3: What model did you use? Include the loss plot. What results did you get (overall accuracy, and confusion matrix).

- Step 4: Did your method work? Include some qualitative results (i.e. Kitti test images, with overlayed 'Car' bounding boxes) to support your claim. What was your overall average IoU?

- Discussion: Discuss the performance of your system. Was it as expected? What challenges did you experience, and how did you overcome them?

4. Submission

The submission should all of the elements described in Section 2.

All deliverables should be compressed into a single **`<zzz>.zip`** `file,` where filename **`<zzz>`** is replaced with your student number. If you are in a team of 2, then concatenate both student numbers, separated by an underscore. The zipped directory should include your code and scripts, your trained parameter files, and all output plots and images.

The report should include a title (e.g. minimally ELEC 475 Lab 4), your name and student number. If you are working in a team of two, then include the information for both partners in the report (but only make one submission in OnQ).

Your code should execute by entering the syntax provided in your included scripts (e.g. **`step3_train.txt`**`, etc.)` on a PyCharm terminal.

The marking rubric is as follows:

Item	mark
Step 3	2
Step 4	3
Report	1
Correct submission format	1
Total:	**7**