BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS

Faculty of Electrical Engineering and Informatics
Computer Vision Systems

Development of custom 2D object detection using Detectron2 library and CARLA simulator dataset

Author: Saif Albuhayder

Consultant: Márton Szemenyei

Budapest, 2022. 05. 25.

Table of contents

1.Overview:

The main task is to develop an algorithm that can reliably detect and track all relevant objects around the vehicles, The purpose for doing this project is to train, test and detect the objects with Faster R-CNN model using the CARLA simulator dataset. And our own image annotation.

In addition, the GPU resources to train the model requires huge memory and this is a constraint for using free COLAB version which provides TESLA T4 cores of size 15GB for a runtime of 10- 12 hours per day. Hence, we limited to run for 2k iterations only and investigated the mean Average precision mAP@0.5 (which is IOU 0.5).

2.Object Detection

Object detection deals with the localization and classification of objects contained in an image or video, and it comes down to drawing bounding boxes around detected objects. It differs from the Image classification which basically sends an entire image through a classifier (such as a CNN), and it gives out a tag associated with a label, but clearly, they do not give any indication on where this tag might be in the image. the state-of-the-art object detection methods can be categorized into two main types: One-stage vs. two-stage object detectors.

In general, deep learning-based object detectors extract features from the input image or video frame. An object detector solves two subsequent tasks:

- Find an arbitrary number of objects (possibly even zero), and
- Classify every single object and estimate its size with a bounding box.

To simplify the process, you can separate those tasks into two stages. Other methods combine both tasks into one step (single-stage detectors) to achieve higher performance at the cost of accuracy.

3. R-CNN – Region-based Convolutional Neural Networks

Region-based convolutional neural networks or regions with CNN features (R-CNNs) are pioneering approaches that apply deep models to object detection. R-CNN models first select several proposed regions from an image (for example, anchor boxes are one type of selection method) and then label their categories and bounding boxes (e.g., offsets). These labels are created based on predefined classes given to the program. They then use a convolutional neural network to perform forward computation to extract features from each proposed area.

In R-CNN, the inputted image is first divided into nearly two thousand region sections, and then a convolutional neural network is applied for each region, respectively. The size of the regions is calculated, and the correct region is inserted into the neural network. It can be inferred that a detailed method like that can produce time constraints. Training time is significantly greater compared to YOLO because it classifies and creates bounding boxes individually, and a neural network is applied to one region at a time.

In 2015, Fast R-CNN was developed with the intention to cut down significantly on train time. While the original R-CNN independently computed the neural network features on each of as many as two thousand regions of interest, Fast R-CNN runs the neural network once on the whole image. This is very comparable to YOLO's architecture, but YOLO remains a faster alternative to Fast R-CNN because of the simplicity of the code.

At the end of the network is a novel method known as Region of Interest (ROI) Pooling, which slices out each Region of Interest from the network's output tensor, reshapes, and classifies it. This makes Fast R-CNN more accurate than the original R-CNN. However, because of this recognition technique, fewer data inputs are required to train Fast R-CNN and R-CNN detectors.

4. Detectron2

Detectron2 is a popular PyTorch based modular computer vision model library. It is the second iteration of Detectron, originally written in Caffe2. The Detectron2 system allows you to plug in custom state of the art computer vision technologies into your workflow. Quoting the Detectron2 release blog:

Detectron2 includes all the models that were available in the original Detectron, such as Faster R-CNN, Mask R-CNN, RetinaNet, and DensePose. It also features several new models, including Cascade R-CNN, Panoptic FPN, and TensorMask, and we will continue to add more algorithms. We've also added features such as synchronous Batch Norm and support for new datasets like LVIS.

5. Code implementation:

For Training we o prepare our data set and the first step is to annotate it, then register it to COCO format and then use a pre trained model to train our custom datset.

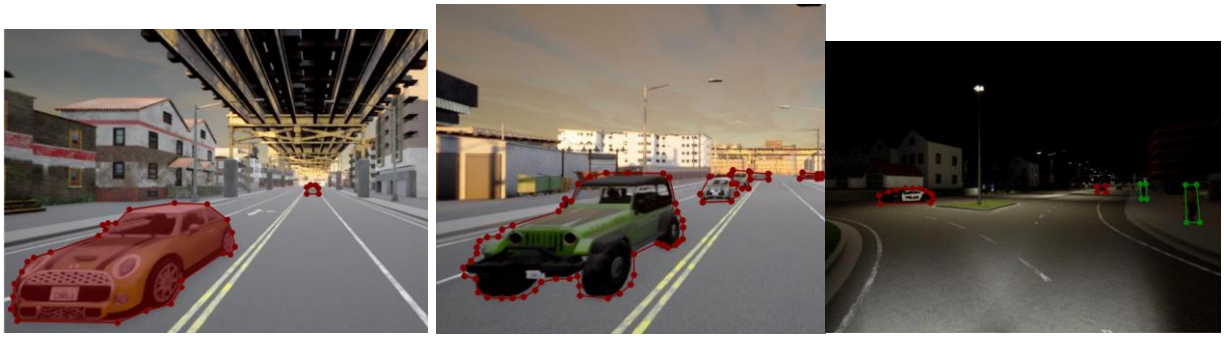
Object Categories used in dataset

Class
pedestrian
bicycle
car
motorcycle
truck

Labels in the CARLA simulator dataset

5.1 Label the Dataset with LabelMe

Accurately labeled data is essential to successful machine learning, and computer vision is no exception. Labelme is a graphical image annotation tool inspired by http://labelme.csail.mit.edu.
It is written in Python and uses Qt for its graphical interface.
Total annotated images is 1200, Total instances is 3826



Dataset example of bbox detection

5.2 Split the data set into train – validation:

We need to Separate data into training and testing sets is an important part of evaluating data mining models. Typically, when you separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing.

After splitting the dataset we needed to make a small “ImagePath” modification on JSON file because of the path changing problem, so made it automatically by implementing this code.

```python
import os
import re
import json
os.listdir('D:\l')
files = [f for f in os.listdir('.') if re.match(r'[0-9]+.*\.json', f)]
files.sort()
len(files)
# print(files)
c = 89;
my_counter = 0
for i in files:
    my_counter += 1;
    c += 1
    a = f"{c}"
    with open(i, 'r') as file:
        print(i)
        json_data = json.load(file)
        json_data['imagePath'] = f"1 ({a}).jpg"
        output_filename = f'{i}'# + '_output.json'
    with open(output_filename, 'w') as file:
        json.dump(json_data, file, indent=2)
# print(my_counter)
```

JSON file editor

We have used 75% train to 25% val. Split

5.3 Register dataset to COCO Format:

To let detectron2 know how to obtain a dataset, we need to implement a function that returns the items in our dataset and then tell detectron2 about this function.

```python
    record = {}
    record["image_id"] = id        #           solved

    filename = os.path.join(directory, img_anns["imagePath"])
    height, width = cv2.imread(filename).shape[:2]  # make it easier for
    record["file_name"] = filename
    record["height"] = height
    record["width"] = width

    print(record["image_id"])
    annos = img_anns["shapes"]
    objs = []
    for anno in annos:
        px = [a[0] for a in anno['points']]  # x coord
        py = [a[1] for a in anno['points']]  # y-coord
        poly = [(x, y) for x, y in zip(px, py)]  # poly for segmentation
        poly = [p for x in poly for p in x]

        obj = {
            "bbox": [np.min(px), np.min(py), np.max(px), np.max(py)],
            "bbox_mode": BoxMode.XYXY_ABS,
            "segmentation": [poly],
            "category_id": classes.index(anno['label']),
            "iscrowd": 0
```

5.4 Train the model:

Choose detection algorithm

We chose our model based on etectron2 model zoo parameters. In this link https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md

Faster R-CNN:

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50-C4	1x	0.551	0.102	4.8	35.7	137257644	model | metrics
R50-DC5	1x	0.380	0.068	5.0	37.3	137847829	model | metrics
R50-FPN	1x	0.210	0.038	3.0	37.9	137257794	model | metrics
R50-C4	3x	0.543	0.104	4.8	38.4	137849393	model | metrics
R50-DC5	3x	0.378	0.070	5.0	39.0	137849425	model | metrics
R50-FPN	3x	0.209	0.038	3.0	40.2	137849458	model | metrics
R101-C4	3x	0.619	0.139	5.9	41.1	138204752	model | metrics
R101-DC5	3x	0.452	0.086	6.1	40.6	138204841	model | metrics
R101-FPN	3x	0.286	0.051	4.1	42.0	137851257	model | metrics
X101-FPN	3x	0.638	0.098	6.7	43.0	139173657	model | metrics

Train the model

After choosing the right model we can start the training with this code:

```
# Create a configuration and set up the model and datasets
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("category_train",)
# cfg.DATASETS.TEST = ("category_train",)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml")
```

Results :

```
[05/24 20:25:15 d2.utils.events]:  eta: 0:00:58  iter: 4919  total_loss: 0.245  loss_cls: 0.06413  loss_box_reg: 0.1589  loss_rpn_cls: 0.0009127
[05/24 20:25:30 d2.utils.events]:  eta: 0:00:44  iter: 4939  total_loss: 0.194  loss_cls: 0.06282  loss_box_reg: 0.1263  loss_rpn_cls: 0.001295  l
[05/24 20:25:45 d2.utils.events]:  eta: 0:00:29  iter: 4959  total_loss: 0.2075  loss_cls: 0.05598  loss_box_reg: 0.1395  loss_rpn_cls: 0.0005623
[05/24 20:25:59 d2.utils.events]:  eta: 0:00:14  iter: 4979  total_loss: 0.2154  loss_cls: 0.06319  loss_box_reg: 0.1319  loss_rpn_cls: 0.0003703
[05/24 20:26:16 d2.utils.events]:  eta: 0:00:00  iter: 4999  total_loss: 0.2279  loss_cls: 0.06457  loss_box_reg: 0.1541  loss_rpn_cls: 0.0003638
[05/24 20:26:16 d2.engine.hooks]: Overall training speed: 4998 iterations in 1:00:47 (0.7299 s / it)
[05/24 20:26:16 d2.engine.hooks]: Total training time: 1:00:53 (0:00:05 on hooks)
```

Results after 5k iteration completed

Sample predictions by the model is shown below.


pedestrian 95%


car 100%


pedestrian 97%
car 100%


pedestrian 95%
car 99%


car 100%
car 99%


car 100%
car 97%
pedestrian 96%

5.5 Metrics:

Mean average precision is the most common metric value which is used for Object detection , it is like average precision is calculated by taking the mean AP over all classes. The true positives and false positives are totally depended on the IOU threshold. Instance if we give the IOU threshold as 0.7 and your IOU is 0.8 then it is considered as True positive . On the other hand, if the IOU is less than 0.7 it is considered as false positive . That's why the ideal IOU threshold is chosen after wise consideration .

mAP doesn't mean it was average of precision , here average precision means finding the area under precision recall curve and is averaged over all categories to find mAP

Precision :

How accurate your predictions are correct :

$$Precision = \frac{\sum TP}{\sum TP+FP}$$

TP – True Positive (predicted as positive it was correct)
FP - False Positive(predicted as positive it was incorrect)

Recall:

It measures how well you find all positives ;

$$Recall = \frac{\sum TP}{\sum TP+FN}$$

TN : True Positive (predicted as positive and was correct)
FN : False Negative (Failed to predict an object it was present)

All the results loges can be found under the link
Results link : https://github.com/sluvd/CV_homework

Pascal VOC metric defines the mAP metric using a single IoU threshold of 0.5
COCO metrics defines several mAP metrics using different thresholds.

Average Precision (AP)

- mAP@IoU= .50:.05:.95
- Primary challenge metric
- mAP@IoU= .50
- PASCAL VOC metric
- mAP@IoU= .75
- Strict metric

Average Recall (AR)

- mAR@max=1
- 1 detection per image
- mAR@max=10
- 10 detections per image
- mAR@max=100
- 100 detections per image

We did the training on all videos and then then compared it to signal video results

Average Precision	000	001	002	003	004	005	006	007	All data
mAP@IoU =.50:.05:.95	0.532	0.578	0.707	0.582	0.493	0.533	0.472	0.471	0.506
mAP@IoU = .50	0.886	0.990	0.965	0.937	0.847	0.990	0.834	1.000	0.864
mAP@IoU = .75	0.567	0.702	0.832	0.645	0.544	0.554	0.528	0.432	0.551
Average Recall									
mAR@max=1	0.302	0.575	0.340	0.250	0.291	0.536	0.420	0.435	0.308
mAR@max=10	0.579	0.653	0.742	0.640	0.552	0.620	0.551	0.537	0.563
mAR@max=100	0.579	0.653	0.742	0.641	0.553	0.620	0.551	0.537	0.565

6.Conclusion:

Object detection is an important research topic. Due to the need of making self-driving cars and autonomous vehicles safer and more reliable. We were able to test, train and detect, As a initial idea to investigate the model performance on detecting the objects in datasets using Faster R-CNN model has resulted in good performance.

References

[1] LabelMe annotation tool - https://github.com/wkentaro/labelme

[2] Object Detection in 2022: https://viso.ai/deep-learning/object-detection/

[3] Detectron2: A PyTorch-based modular object detection library - https://ai.facebook.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library-

[4] Detectron2 GitHub repository - https://github.com/facebookresearch/detectron2

[5] Detectron2 Model Zoo - https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md

[6] Detectron2's documentation - https://detectron2.readthedocs.io/en/latest/index.html

[7] COCO Detection Evaluation - https://cocodataset.org/#detection-eval

[8] CARLA simulator - https://carla.readthedocs.io/en/latest/start_quickstart/