Special Topics - MSAI 495

Detailed Project Plan

“On-road object detection for self-driving cars”

February 21st, 2022

Professor
Reda Al-Bahrani

Students
Ayushi Mishra
Preetham Pareddy
Ana Cheyre

Objective Description

Self driving cars might be a luxury in the current world but the technology is rapidly changing to be available for everyone. They are useful because ideally, machines don't make the mistakes humans do, so there would be less accidents like crashes. It will also save a lot of time for humans, making their life much more efficient. Computer vision is a critical component for the functioning of such cars. There is an absolute need to create better models that minimize, if not completely eradicate, danger on the road. This is the reason we chose to work on object detection on the road. The goal of this project is to develop a computer vision model that solves some of the multiple tasks that the system of self-driving cars must execute to be able to drive autonomously. This refers to the detection of diverse elements found on the streets so that the car is able to identify if this type of obstacle is on the road or not.

Related Work/Literature Survey

Object Detection has always been a burning issue in computer vision which is applied in many areas such as security, surveillance, autonomous vehicle systems, and machine inspection. The popular object detector algorithms implemented in all these domains are either region-based detection algorithms or single-shot detection algorithms.

Baseline

The paper that strongly influences our project works and forms a baseline for us is the R-CNN (Region based Convolutional Neural Network) family of papers by a group of researchers at UC Berkeley (Ross Girshick et al.). In the R-CNN family of papers that we are referring to, the evolution between different versions mentioned in the papers was usually in terms of computational efficiency (incorporating the various training stages), decreasing testing time, and improving the model performance (mAP).

- **R-CNN:-** Rich feature hierarchies for accurate object detection and semantic segmentation (https://arxiv.org/abs/1311.2524)

  In this paper, the issue of object detection is resolved by introducing a concept called R-CNN. The network comprises of:
  a. A region proposal algorithm that is responsible for generating “bounding boxes” or locations of potential objects within the image.
  b. A feature generation stage that is responsible for obtaining features of these identified objects, with the help of a CNN.
  c. A classification layer that is used to predict which class the object belongs to.
  d. A regression layer that is used to make more precise coordinates of the object bounding box.

- **Fast R-CNN:-** Fast R-CNN (https://arxiv.org/abs/1504.08083)

  One year after the original R-CNN paper the authors published this paper to build upon their previous work. Since R-CNN was slow, hard to train and had a large memory requirement; with fast R-CNN the authors combined the three different parts that we had in the R-CNN system (a CNN, SVM, Bounding Box Regressor) into one seamless architecture. *The fast R-CNN was found to train the VGG16 network 9 times faster than their previously developed R-CNN.* But the amazing thing about this system is that the inference is 213 times faster and achieves a higher mAP. This introduced one seamless end-to-end system that could be trained with back-propagation.

- **Faster R-CNN:-** Faster R-CNN: Towards Real-Time Object Detection with Regional Proposal Networks (https://arxiv.org/abs/1506.01497)

  The Faster R-CNN architecture consists of the RPN(another convolutional network) as a region proposal algorithm and the Fast R-CNN as a detector network. This not only decreased the region proposal time from 2s to 10ms per image but also allowed the region proposal stage to share layers with the following detection stages, resulting in an overall significant improvement of feature representation.

Main approach

With Faster R-CNN as our baseline we looked at other state of the art approaches and found a paper "*YOLOP: You Only Look Once for Panoptic Driving Perception*" by Dong Wu, Manwen Liao, Weitian Zhang, and Xinggang Wang (https://arxiv.org/pdf/2108.11250v6.pdf).

In this paper, the authors put forward a brand-new, simple and efficient network, which can simultaneously handle three driving perception tasks of object detection, drivable area segmentation and lane detection and can be trained end-to-end. The model performs exceptionally well in achieving or greatly exceeding state-of-the-art level on all three tasks. And it is the first to realize real-time reasoning on embedded device Jetson TX2, which ensures that the network can be used in real-world scenarios.

With 45 frames per second, YOLO establishes its superiority over other object detection algorithms by being orders of magnitude faster than them. The YOLO algorithm's only limitation is its inability to detect small objects within the image with great clarity due to spatial constraints. For eg- It may struggle in detecting a flock of birds.

Project extension

We will closely follow the approaches mentioned in the above papers. However when it comes to our main approach we will tweak the state of the art methods by using different methods like warmup and cosine annealing on the learning rate of the model to create effective learning rate schedules. We also hope to explore the possibility of integrating the R-CNN modeling techniques with YOLOP.

Dataset and Pre-processing

Dataset

The BDD100K database is the largest and most diverse open driving video dataset so far for computer vision research, it contains around 120 million images of roads, with signs and different objects on them, obtained from 100,000 videos on roads where each video is about 40 seconds long, 720p, and 30 fps. The data were collected from diverse locations in the United States, also covering different weather conditions, including sunny, overcast, and rainy, as well as different times of the day including daytime and nighttime, giving it a great diversity of different environments. Because of this, the algorithm trained on the BDD100K dataset will be robust enough to migrate to a new environment. The BDD100K dataset consists of three dataset parts: training set with 70K images, validation set with 10K images, and test set with 20K images.

For object detection, each image has object bounding boxes, and these boxes contain the respective label. It helps to understand the distribution of the objects and their locations. The images contain labels for 10 different types of objects, which correspond to 10 different classes: traffic light, traffic sign, car, person, bus, truck, rider, bike, motor and train.

Class	Frequency
Traffic light	265,906
Traffic sign	343,777
Car	1,021,857
Person	129,262
Bus	16,505
Truck	42,963
Rider	6,461
Bike	10,229
Motor	4,296
Train	179

Frequency of classes in BDD100K database

The database is divided into images with a respective ID, and another folder with a JSON file for each dataset (train, validation, test) indicating the labels of the objects and other information of the image, like timestamp, scene description, time of day, coordinates of box location, etc.

Here we can see an example of an image and a fragment of its respective JSON:

Image ID: "0000f77c-6257be58.jpg”

JSON:


```
{
   "name": "0000f77c-6257be58.jpg",
   "attributes": {
     "weather": "clear",
     "timeofday": "daytime",
     "scene": "city street"
   },
   "timestamp": 10000,
   "labels": [
    {
      "id": "1",
      "attributes": {
        "occluded": false,
        "truncated": false,
        "trafficLightColor": "NA"
      },
      "category": "car",
      "box2d": {
        "x1": 49.44476737704903,
        "y1": 254.530367,
        "x2": 357.805838,
        "y2": 487.906215
      }
    }, (...)
   ]
 }
```


Pre-processing

To improve the performance of our model, we will need to preprocess the data using different methods.

First, we have to process the images files, both resizing and converting them into pixel arrays in order to be able to work with them in the models. The input size of the images is 1280×720×3, and we will resize them to 640×384×3 that will allow the model to run faster, but still keep all the

information needed for model training. Then, to convert the images to pixel arrays, we'll use the *Pillow* library, which takes an image and returns the content as an object containing pixel values, which can be transformed into a numpy array. Each array contains 3 channels, which represent the colors red, green and blue from "RGB" color code, and the value of each pixel between 0 and 255 with respect to each color channel.

Second, we will use data augmentation to increase the variability of images and make our model robust in different environments. Photometric and geometric distortions will be used to create more data. For photometric distortions, we can adjust the hue, saturation and value of images. On the other hand, to process images to handle geometric distortions we plan to use random rotating, scaling, translating, shearing, and left-right flipping. To carry out these transformations, the *TorchVision* library will be used, which has integrated functions that allow these tasks to be carried out straightforwardly.

Methodology

The main algorithm which we try to copy for this purpose is YOLOP since it's considered the state of art for the task of object detection on the road. It's a panoptic driving perception network to perform traffic object detection, drivable, area segmentation and lane detection simultaneously. Though our main goal is object detection, supporting the research we've got done, the opposite tasks (drivable area segmentation and lane detection) help in improving the performance of object detection. Because of the data shared by multitask, the prediction results of YOLOP are more reasonable. For instance, YOLOP won't misidentify objects removed from the road as vehicles. Moreover, the samples of false negatives are much less and also the bounding boxes are more accurate. However, this is dependent on the computing power that's available to us. So our main task is object detection but the drivable area segmentation and lane detection tasks are optional.

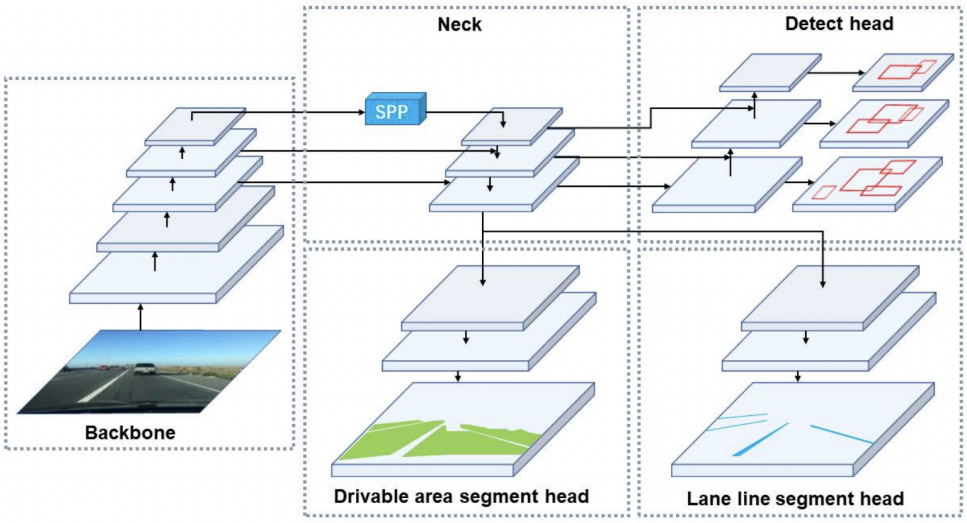
Architecture

YOLOP is composed of one encoder for feature extraction and three decoders to handle the specific tasks. More specifics about each part are given below:

Encoder - The encoder contains a backbone and a neck. The backbone performs the task of extracting features from the image. CSPDarknet is being used for this purpose. The neck is employed to fuse the features generated by the backbone. Our neck is principally composed of the Spatial Pyramid Pooling (SPP) module and therefore the Feature Pyramid Network module.

Decoder - The decoder consists of three heads for the three specific tasks the model is getting used for i.e traffic object detection, drivable, area segmentation and lane detection

- ***Detect head*:** An anchor based multi-scale detection scheme is being employed as the framework for object detection. Initially, Path Aggregation Network (PAN), a bottom-up feature pyramid network, was used. FPN is responsible for transferring semantic features top-down, and PAN is responsible for transferring positioning features bottom-up. Both are then combined to get a higher feature fusion effect, so the multi-scale fusion feature maps within the PAN are used for detection. Then, each grid of the multi-scale feature map are going to be assigned three prior anchors with different aspect ratios, and also the detection head will predict the offset of position and also the scaling of the peak and width, similarly because the corresponding probability of every category and also the confidence of the prediction
- ***Drivable Area Segment Head & Lane Line Segment Head (optional)* :** Both the processes adopt the identical network structure.The bottom layer of feature pyramid network is fed to the segmentation branch, with the dimensions of (W/8, H/8, 256). After three upsampling processes,the output feature map is restored to the dimensions of (W, H, 2), which represents the probability of every pixel within the input image for the drivable area/lane line and therefore the background. due to the shared SPP within the neck network, no extra SPP module is added to segment branches. Additionally, the closest Interpolation method is employed within the upsampling layer to scale back computation cost rather than deconvolution. As a result, not only do the segment decoders gain high precision output, but even be in no time during inference.



YOLOP architecture

Loss

Since there are three decoders for the three tasks that the model is performing, the loss function consists of three components as shown below

$$\mathcal{L}_{det} = \alpha_1 \mathcal{L}_{class} + \alpha_2 \mathcal{L}_{obj} + \alpha_3 \mathcal{L}_{box}$$

where

L_{class} and L_{obj} : focal loss, which is utilized to reduce the loss of well-classified examples, thus forcing the network to focus on the hard ones.
L_{class} - penalizing classification
L_{obj} - confidence of one prediction.
L_{box} is L_{CIoU} - takes distance, overlap rate, the similarity of scale and aspect ratio between the predicted box and ground truth into consideration.

Training structure

Input: Target neural network F with parameter group:
Θ = {θenc, θdet, θseg}
Training set: T
Threshold for convergence: thr
Loss function: L_{all}
Output: Well-trained network: F(x; Θ)

1: procedure TRAIN(F, T)
2: repeat
3: Sample a mini-batch (xs, ys) from training set T .
4: ` ← Lall(F(xs; Θ), ys)
5: Θ ← arg minΘ `
6: until ` < thr
7: end procedure
8: Θ ← Θ\{θseg} // Freeze parameters of two Segmentation heads.
9: TRAIN(F, T)
10: Θ ← Θ ∪ {θseg} \ {θdet, θenc} // Freeze parameters of Encoder and Detect head and activate parameters of two Segmentation heads.
11: TRAIN(F, T)
12: Θ ← Θ ∪ {θdet, θenc} // Activate all parameters of the neural network.
13: TRAIN(F, T)
14: return Trained network F(x; Θ)

Though the model is state of art we want to tweak it in certain ways for experimentation. We want to try different methods like warmup and cosine annealing on the learning rate of the model to create effective learning rate schedules. Apart from this we want to research R-CNN further to see if we can incorporate some techniques used in that model with the YOLOP.

The metrics used for the object detection task would be:

Precision (P) and recall (R) for each object category, in this case we will be particularly interested in the amount of false negatives (looking for high recall), because in a self-driving car there is no space for errors because it can risk people's lives. Also the mean average precision (MAP) which is a very common metric in object recognition as it measures how good the model is at performing the classification of objects. Finally, we will keep track of accuracy and the training loss after each iteration.

Like we have mentioned before, we are focusing on recall which is a common norm in traffic object detection. The top models for this dataset are:

YOLOP	89.2
YOLOv5s	86.8
Multinet	81.3
Faster R-CNN	77.2

We will measure the performance of our neural network by measuring it against the above existing models. As a baseline, we will attempt to outperform Faster R-CNN. Recent state of the art models have improved this score by at least 10% Thus, we will attempt to reach a recall of 87%. Additionally, other object detection specific evaluation metrics we will consider are multiple object tracking precision, IoU (intersection over union), multiple object tracking accuracy, and mean average precision with heading.

We intend to investigate, and potential apply the following methods to aid in improving accuracy and precision:

- ***Implementing different sampling strategies:*** Since point clouds are usually sparse, by sampling we would be able to improve the data imputed to the NN.
- ***Weighted boxes fusion:*** From various object detection models ensembling boxes.
- ***Restricting our training dataset to high visibility and minimal occlusion frames:*** Using frames without occlusions/better visibility and good bounding box info can improve training. Using subsampling, we can choose the frames that give the best visibility with respect to the bounding box.

Computational Problem

Since we do not have a lot of time and computational power to train, we want our architecture to perform well with less training. Therefore, we will explore methods expected to give us better inference. We plan to start with existing state of the art architectures, then propose a novel model by modifying these models.