Artificial Intelligence Applications in Autonomous Vehicles: Training Algorithm for Traffic Signs Recognition

C Dursun[1], T I Erdei[1], G Husi[1]

[1] Mechatronics Department, University of Debrecen, Faculty of Engineering Debrecen, Hungary

candrsun@gmail.com

Abstract. As the last point of autonomous vehicle development, these vehicles are being evolved by most of the car manufacturing companies, which increases the need for novel image recognition and automation solutions. This paper presents autonomous driving for vehicles at the stage of level 2 (partial Automation; the vehicle can perform steering and acceleration, though the human driver still monitors all tasks and can take control at any time). YOLO (You Only Look Once), is a Python-based image processing algorithm that was used to achieve the goals. This algorithm is extremely useful due to its real-time capabilities. A new data set was gathered using miniature signs, and a Python script was developed to label these newly trained objects. The algorithm draws bounding boxes around the object with object name and the rates of accuracy. The created system performs well, with further plans to improve it, and apply it in a simulated/modelled environment.

1. Introduction

The world automotive engineers evaluate autonomous driving at six levels, "0" is completely under the control of driver without any assist, "1" the vehicle features a single automated system, for instance, controlling the speed by cruise control, "2", the vehicle can control the steering wheel and the acceleration, but the human can monitor all the tasks and takes the control if needed. Levels "3-6" are the automated systems which monitor the driving environment [1].

The algorithm You Only Look Once (YOLO) is an image processing algorithm which is very popular lately. Although other algorithms are very accurate, none of them is as fast and precise as YOLOv3 as it can process images in real-time [2]. In this article, I am going to present how to collect custom data to create a dataset and implement it for YOLOv3 algorithm.

2. Background

2.1. Understanding the YOLO algorithm

The COCO (Common Objects in Context) is large-scale object detection, segmentation, and captioning dataset [3]. YOLO is much faster compared to other algorithms [4]. We can make changes in accuracy or speed, but if we increase the speed-accuracy will decrease, and vice-versa. R-CNN (Convolutional Neural Network) based object detection algorithm primarily determines the areas where objects are likely to be found then in these areas, processed for classification [5]. Although this method gives good results, one image is processed under two different operations which increases the count of the process and as a result gives a low FPS (Frames per second).

On figure 1, the structure of R-CNN can be seen. R-CNN and Faster R-CNN algorithms can process the image in one operation, but when it comes to real-time image processing, the algorithm cannot perform more than 7 FPS. YOLO can estimate the classifications and the coordinates by crossing images to deep convolution neural network to detect the object in real-time which can reach more than 40 FPS with an Nvidia CUDA supported GPU (Graphical Processing Unit) [7].

Namely principal of the estimate is recognition an image in a single process. To do this process firstly, it separates the image into an SxS (S number by S number) grid. But this also can be 3x3, 5x5, 19x19 etc. [8]. According to this process output of the image can be seen in the following image.

Each grid inspects if there is an object inside them, if yes then check if the object midpoint of the grid, if yes again then check its length, height and find which class it belongs to.
To be more specific, for example, in the presented image, 7th grid is responsible for recognition because the midpoint of the car is on the 7th grid.

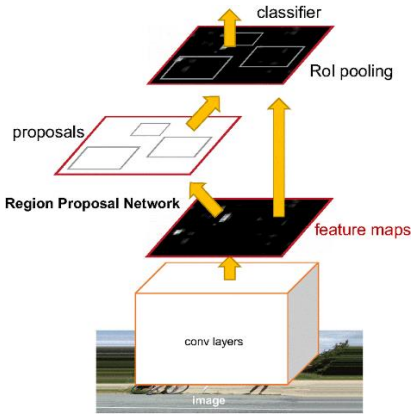

Figure 1. R-CNN Architecture [6]

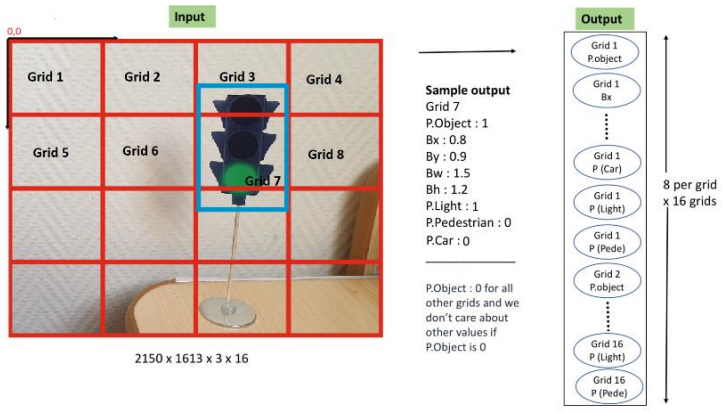

Figure 2. YOLO algorithm input-output

According to this YOLO creates prediction vector for each grid and inside of them: Confidence score: This score presents when the object inside the grid and its accuracy, at "0" it's absolutely outside of the grid, at "1", absolutely inside. The operating parameters are: Bx: X coordinate of the object's midpoint; By: X coordinate of the object's midpoint; Bw: Width of the object; Bh: Height of the object.

Probability of classes: Algorithm estimate of all classes. For instance, in the presented image when we look to grid 7, if there is a car, car would be "1", light would be "0" and pedestrian also "0". Calculation for confidence score (IOU = intersection over union, the value that is estimated between boxes): confidence score = P(Object) x IOU. P(Object). This tells us if "P" is within the grid. [9].

According to output vector each grid can recognize only one object. For instance, if we use 3x3 grid YOLO could recognize 9 objects. Well, if there are more than one object in a grid what would happen? Or even midpoint of two object is inside of one grid. This was the one of the problems that could not be answered with the first version of YOLO. With the second version of YOLO algorithm was extended with Anchor Boxes which solves this issue [10]. The Anchor Boxes method was firstly used on R-CNN algorithm and it recognizes the boxes around the object. With two anchor boxes, each object in the training image is assigned to a grid cell that contains object's midpoint and anchor box for the grid cell with highest IOU [11].

After adding the anchor boxes, the output vectors can be mathematically expressed. In figure 3, two anchor boxes can be seen. In this image midpoint of the Traffic light and the firefighting truck are exactly inside the same grid. Our goal is to draw a bounding box around the object and at the same time know the height and the width of the boxes.

With the second version of the YOLO we have the anchor boxes we can directly find the anchor box that is most similar to the object box and we can estimate the width and height difference of that box relative to the anchor box, so we can handle the problem that we have faced with the first version of

YOLO. Now we can predict Tx, Ty, Tw, and Th (where Tx, Ty, Tw, and Th are the distance from the given grid border on each axes) for every box by neural network. Thus, we know which grid cell we are working on which is expressed by Cx and Cy (where Cx and Cy are the distance from the top left point, expressed in X, Y coordinates.) Finally, the last items are Pw and Ph which represent the height and width of anchor boxes. On figure 4, we can observe as the dotted rectangle represents the anchor while, and the blue box the predicted boundary box.

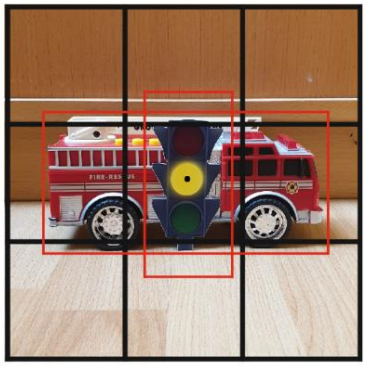

Figure 3. Anchor boxes

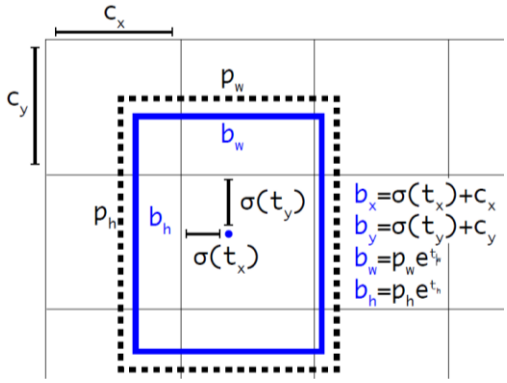

Figure 4. Anchor boxes prediction [12]

In this presented method we are normalizing the parameters between "0" and "1" as a result our neural network is being more stable which makes the training process easier and precise. In the following section data collection for YOLO v3 will be introduced.

3. Dataset training and testing

3.1. Collecting data for training

The purpose of the project is training the algorithm for autonomous driving therefore we are going to collect traffic sign (speed limit and stop sign) and traffic light images for identification. One of the feature plans of project is implementing YOLO v3 real-time image processing algorithm into a robot for a real-world demonstration. A track line will be drawn on the floor, then the miniature traffic signs, and traffic lights will be placed around the track. The robot will follow the line, and YOLO will check the signs and make decisions (for example, control speed based on speed limit signs). The traffic signs must be captured for a custom dataset as the miniatures slightly differ from full-sized ones.

Traffic sign figures were found online and printed out. The traffic sign and traffic light figures are presented on the following image. We need to provide hundreds of images that will be used to train new detection classifier, YOLO v3. Many pictures were taken in different conditions, to teach the algorithm different disturbances it may encounter on an image. These are as follows:

- All photos must be in 1:1 scale
- Different angles (changing the position of the figure after each capture)
- Random backgrounds (putting the figure to different background after each capture.)
- Different light sources (capturing photos under sunlight or in a low light source.)
- Different visibility (for example half of the figure is visible while half of it not
- Capture the signs together in one photo and separately

As seen on figure 6, one sign was captured front of completely different backgrounds. I took pictures of them in one area then moved them to a different area and repeated the process in harder and harder visual environments with more random objects and busier backgrounds. I tried to include similar objects in some pictures to help the detector avoid falsely identifying things that look like traffic signs. In total, 700 images were taken (100 per sign).

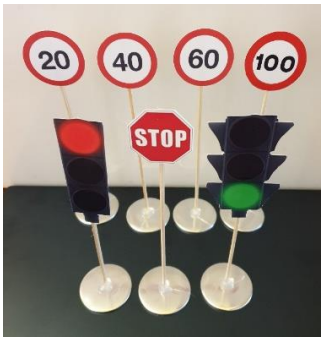

Figure 5. Traffic signs and traffic lights for training

Figure 6. Capturing pictures with different backgrounds

3.2. Image labelling (Annotation), model training and testing

After collecting the images next step is labelling every single image one by one to train, what the objects are and where they are located. The "Labelimg" free labelling tool was used, which is a graphical image annotation tool. With the tool all that was needed is opening the training directory click the "create Rectbox" button and draw a rectangle around the objects in the image, then type in a label for each. The saved XML file contains the size of the image and where the labelled object is using an X-Y coordinate system.

Moreover, we need to create CLASSES.txt and CLASSES_annotation.txt files before starting to train the model. Classes exact meaning is separating object from each other for instance, "traffic light green" or "speed limit 100" As CLASSES.txt presented on the following image, it contains name of the classes while CLASSES_annotation.txt contains location of the image and coordinates of the bounding boxes for every single pictures that we collected. As for content of CLASES_annotation.txt, we are inserting each XML files content into a single TXT file. This concludes the model training.

The YOLO v3 algorithm is a one-shot learner which means that to predict the object, it sends each image to the network only once. To train the model we will use Nvidia GTX 1070 GPU with 8gb of ram, because YOLO can use the full potential of CUDA cores. An important point before start training is to be sure there is no space between any folder name for instance and of there are, they must be substituted with underscores. It took 10 hours to complete the training process. At this time final trained weight is also created, which we are going to use to test the model in the next step.

To test the model, we used a webcam which is connected to a PC. As a result, model is working successfully, and can predict the traffic signs and lights in real time, performing at about 13-14 FPS, which is not too fast, but satisfactory for our model. Detecting object at a long distance is at times unreliable, but the system works well otherwise, even in low-light conditions as seen on figures 7-8.

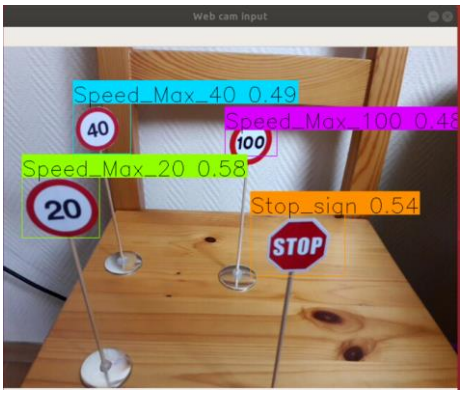

Figure 7. Testing the model under well-lit conditions

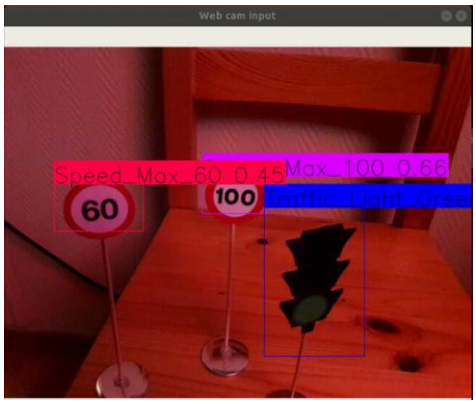

Figure 8. Testing the model in low-light conditions

4. Conclusion

YOLO v3 has demonstrated remarkable performance gains while running at real-time detection. Results can be used many applications if it can be trained with a proper dataset. I classified 7 classes for an existing image recognition algorithm called YOLO to use it on my future project which will be detailed on the future plain section. I faced many problems during the implementation. For example, all the modules had to be compatible with each other and its very time consuming to find the current versions.

5. Future plans

The focus of further developments would include more traffic signs, increasing the efficiency of the model using a loss function, and implementing the model onto a line following robot for a real-world demonstration. There would be miniature traffic signs and traffic lights around the track. While the robot follows the line, model will check the traffic signs then makes decisions. For instance, if algorithm recognizes “max speed 50” sign then robot is going to limit the speed to “50” percent, if recognizes “max speed 30” sign then robot’s speed will be “30”. When model recognizes stop sign or red traffic light will stop immediately until traffic light turns to green.

References

[1] "iotforall.com," Autonomous driving levels, [Online]. Available: https://www.iotforall.com/5-autonomous-driving-levels-explained/.

[2] "YOLOv3 Official Website," [Online]. Available: https://pjreddie.com/darknet/yolo/.

[3] "COCO (Common Object in Context," [Online]. Available: http://cocodataset.org/. [Accessed 03 03 2020].

[4] "Yolo vs others," [Online]. Available: https://medium.com/analytics-vidhya/everything-you-need-to-know-to-train-your-custom-object-detector-model-using-yolov3-1bf0640b0905.

[5] "CNN Convolutional Neural Network," [Online]. Available: http://cs231n.github.io/convolutional-networks/.

[6] G. S. Tran, "ResearchGate CNN Architecture," [Online]. Available: https://www.researchgate.net/publication/324549019_Towards_Real-Time_Smile_Detection_Based_on_Faster_Region_Convolutional_Neural_Network.

[7] S. Djahel, "Researchgate," [Online]. Available: https://www.researchgate.net/publication/320592354_A_Novel_YOLO-based_Real-time_People_Counting_Approach#pf1

[8] "Evolution of Object Detection and Localization Algorithms," [Online]. Available: https://mc.ai/evolution-of-object-detection-and-localization-algorithms/.

[9] J. Du, "ResearchGate," [Online]. Available: https://www.researchgate.net/publication/324754316_Understanding_of_Object_Detection_Based_on_CNN_Family_and_YOLO.

[10] A. Ng, "Anchor boxes," [Online]. Available: https://www.coursera.org/lecture/convolutional-neural-networks/anchor-boxes-yNwO0.

[11] Mathworks.com, "Anchor Boxes explained," [Online]. Available: https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html.

[12] J. Hui, "medium.com," [Online]. Available: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.

Acknowledgements

I cannot express enough thanks to my supervisor Mr. Timotei István Erdei for his continued support and encouragement. I am also very grateful to all my teachers who helped me in this project. The research was supported by the University of Debrecen, Doctoral School of Informatics.