2017 2nd International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT), October 23, 2017, Jakarta, Indonesia

Towards Self-driving Car Using Convolutional Neural Network and Road Lane Detector

Brilian Tafjira Nugraha[1], Shun-Feng Su[2]
Department of Electrical Engineering
National Taiwan University of Science and Technology (NTUST)
Taipei, Taiwan, ROC
[1]M10507811@mail.ntust.edu.tw, [2]sfsu@mail.ntust.edu.tw

Fahmizal
Electrical Engineering and Informatics Vocational College
University of Gadjah Mada
Yogyakarta, Indonesia
fahmizal@ugm.ac.id

***Abstract*—The advancement of Computer Vision these days has grown up beyond imagination. Recently, many researchers and tech companies are competing to develop self-driving car using either traditional or deep learning approach. YOLO (You Only Look Once) is one of the real-time CNN methods that aims to detect objects from images. On the other hand, Road Lane Detector is used to detect road track from video's frames and to provide additional information that can be helpful for the decision-making process of the self-driving car. In this paper, we use YOLO as the object detector and polynomial regression as the road guidance in the real-world driving video simulations. We use NVIDIA GTX 1070 with 8 GB of RAM for the computations. The result shows a matching pair between those two methods for self-driving car environment and road lane guidance.**

Keywords—self-driving car; convolutional neural network; yolo; road lane detector

I. INTRODUCTION

Human's eyes are the ultimate environment for them to interact and to guide them to the vast and to the unimaginably big world. Thanks to those gifts, we advance rapidly despite the occurrence of natural disasters and unending war. And now, we are beginning to expand our gifts into the-so-called machine/computer in order to help us automating things for the human benefits, it is called sensor. These gifts are not come for free. People want them to extract so many information out of these, and these demands hail to the born of The Computer Vision.

Computer Vision is one of the area of Artificial Intelligence where they want to extract information out of images. And with the vast demands and expectations of people toward this field, it is rising beyond beliefs: starting from object detection, pattern recognition, action recognition, automatic guidance, and so on. Many papers have been published about it, especially in Deep Learning and Convolutional Neural Network (CNN). The CNN method has largely been exploited in various fields: image processing, machine learning, video analysis, natural language processing, and much more. In the computer vision field, the CNN method has been mainly used for object detection and automation based on the images/videos. GoogLeNet, Faster R-CNN, Single Shot MultiBox Detector (SSD), and You Only Look Once (YOLO) are examples of the succession of the CNN. Those CNN methods are mainly used for object detection and recognition.

GoogLeNet [1], the winner of ILSVRC 2014, uses a deep CNN architecture with 22 network layers which are called as Inception. The core of the architecture is by piling convolutional and pooling layer in a sequential manner. With this high and piling computation demands, the computation time will be its downside where the detection approach that we want to use is it in a real-time environment. Faster R-CNN [2], one of the incremental invoked CNN method that takes advantage of the convolutional network sharing from Fast R-CNN [3], uses Region Proposal Network (RPN) to generate the region of object proposals which will be used by Fast R-CNN for object detection. Even though they already use CNN method in all of the steps to speed-up the computation time, it still could achieve up to 7 frames-per-second (fps). Based on our needs for real-time object detection, these previous two methods are still not applicable for that. SSD [4], on the other hand, tackles the problem, and the overall results are better on the YOLO version 1 [5], but not on the Yolo version 2 [6].

Road Lane Detection is one of the main concern in the application of many self-driving car engineers. There are many approaches that can be used for the edge detection to help the road lane detection: Sobel, Laplacian, and Canny detectors. Sobel edge detection [7] takes advantage of gradient magnitude with the greatest rate's change in the light intensity using two 3x3 kernels in x and y directions. On the other hand, Laplacian edge detection [8] uses only one kernel. The main downside of this detection method is the extreme sensitivity of noises. The Canny detector [9], however, goes one step further by applying the Gaussian filter and the Non-Maximum Suppression (NMS) and thresholds the range of the gradient intensities. Although these methods give additional door to more research, it also put a limitation on the lane detection performance itself since it adds more computation to the model that we built. On the other hand, the computation will be much faster if we can put the edge detection and the lane construction to run in parallel.

In order to make a sustainable self-driving car, we need three main parts: object detection, lane detection, and

978-1-5386-0510-3/17/$31.00 ©2017 IEEE

controller. In this paper, we have combined these parts to help the driver with more info about the object in their driving environment and the road lane guidance. The proposed method combines YOLO version 1 for the object detection, polynomial regression with thresholding as the road lane guidance, and a controller that manages the information between those two systems. The proposed methods can be used for object detection, object position detection (left, front, or right), steering suggestion, and road lane's guidance.

The paper's main body is organized as follows: Section I is the briefing of some backgrounds and the introduction of our proposed methods, Section II and Section III contain the methodologies that will be used in the experiment, Section IV presents the experimental results of the methods, Section V talks about the discussion of several issues that occur on our methods and the limitations that come with it, and Section VI summarizes the conclusion of our proposed methods and possible ideas for the next improvement.

II. Overview of YOLO and Road Lane Detector

A. YOLO

YOLO implements a deep CNN architecture for its object detection approach. It consists of 24 convolutional layers along with 2 fully connected layers, where it is actually inspired by the GoogLeNet model. However, they change the approach from Inception to simply 1x1 reduction layers along with 3x3 convolutional layers. They also have trained a lightweight YOLO for fast object detection where they use less than half convolutional layers along with fewer filters.

The proposed methods use a smaller version of the original YOLO (i.e. YOLO-small) for the object detection's task. It has the same architecture with fewer computation demands. To use YOLO, we need to resize the image input into 448x448x3 pixels, so that it will be able to process the object detection as in their architecture. In the Fig. 1, YOLO feeds the resized images to the convolutional layers with dropout, and inferences the results by computing the fully connected layers. The outputs of the detection object's name, x mid-point coordinate of the object, y mid-point coordinate of the object, the width of the object, the height of the object, and the accuracy of the predicted object. The object detection is pre-trained on the ImageNet competition dataset with 1000 classes [10]. They also use dropout and data augmentation to avoid the overfitting issue. It predicts 98 bounding boxes for each image and class probabilities for each box and presents a fast computation because of its single feed-forward evaluation network. The architecture of the original YOLO is shown in the Fig. 1.

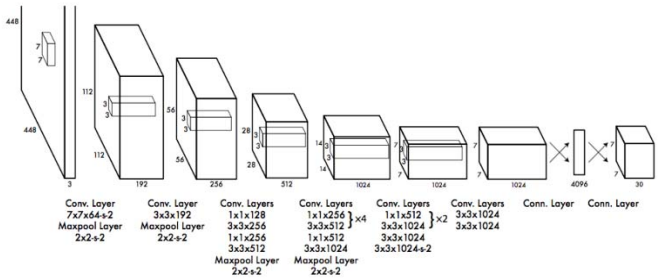

Fig. 1. YOLO's original architecture for the object detection.

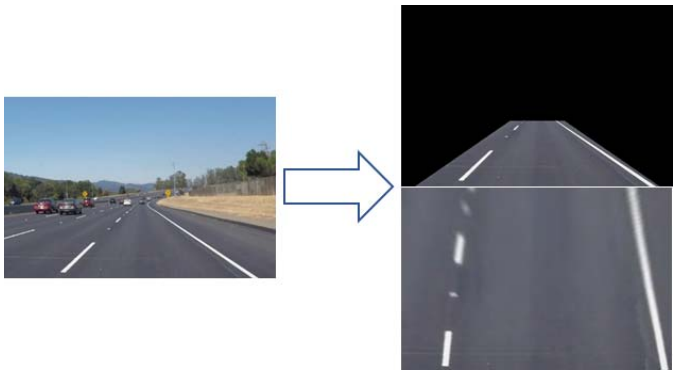

Fig. 2. Image warping process.

B. Road Lane Detector

The road lane detector consists of four main parts: warping, filtering, detecting the road lane, and de-warping. In the warping part, we set The Region of Interest (RoI) of the road lane and change its perspective into new images. Fig. 2 illustrates the process of setting and warping into new images. In this way, it will be much easier for us to visualize and to analyze road lane in detail.

The process of warping the images is handled by changing the perspective of the input images (using *cv2.warpPerspective* from the *cv2* library in *Python*) where it tries to approximate (*M*) the new images' points (*dst*) based on the previous RoI images (x, y) using (1)(2). Next, we do the filtering to separate the lane color with the others. In the filtering process, we pick the range of white and yellow color from the images using *LUV* and *LaB* image formats to get yellow and white colors from the road lane as shown in Fig. 3.

$$div(x,y) = (M_{31}x + M_{32}y + M_{13}) \quad (1)$$

$$dst(x,y) = (M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23})/div(x,y) \quad (2)$$

After the filtering, we detect the road lane points by getting the nonzero values from the previous process. First, we crop the images into 10 sub-images and divide them into left and right. The purpose of this is to get the peak of nonzero values of each lane (left and right). Then, we add the area nearby the peak values to the list of left and right points into the collected points.

Polynomial regression (using the function *np.polyfit* provided by the *numpy* package in *Python*) enables us to approximate the lane/curve (3) by fitting lane points (*y*) that have been collected with coefficient points (*p*) into the best approximated lane points (*Polyfit*) by minimizing the error (*E*) from the fitted lanes (4). So, by getting right and left points, we could make functions to approximate the lane structures by feeding them into the regression method to find the best fit of the road lane detection. Finally, we compute the original perspective by de-warping the new images to the input images as shown in Fig. 4

$$Polyfit(x,y,2) = p^2x + px + p \quad (3)$$

$$E = sigma(Polyfit(x,y,2) - y)^2 \quad (4)$$

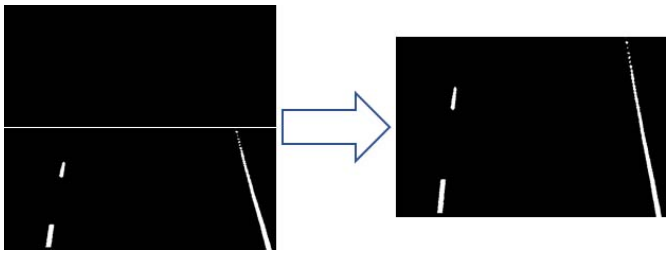

Fig. 3. Filtering process to distinguish between the road points and the background.

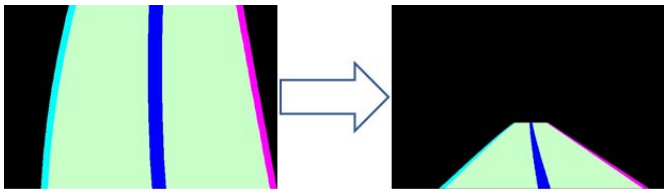

Fig. 4. Road lane detection and de-warping processes.

III. Overview of The Controller

A. Overview

Before we run YOLO and road lane detector, we need to get the information about the general road lane width in the pre-setting configurations to adjust the road width and the offset length. After that, the controller loads the camera/video frames and feeds them into YOLO and Road Lane Detector. After combining both results, it draws the road lane based on the results from the lane detector. Then, it estimates the other object positions by comparing their bounding box coordinates. Then, we compare the coordinates with the road lane coordinates to measure the object positions. Finally, we combine both results into one image for each frame to show the detection results of our proposed methods. The design of the controller with YOLO and road lane detector is shown in the Fig. 5.

B. Pre-Setting Graphical User Interface

The pre-setting configuration uses the road width to calculate the offset position of the vehicle from the center of the road lane. With this configuration, the estimate road offset can be calculated for further purposes. The GUI for the configuration is shown in the Fig. 6.

C. Controller

The controller is used to facilitate the information flows from YOLO and road lane detector. It also computes the object positions from the road lane and updates the road lane and the performance of the system in the real time. We divide the controller function into six different parts:

- YOLO object detection function: call the object detection to find the object positions from the image frames and return the object points (x, y, width, and height) of the detected objects as shown in the Fig. 7 (a).
- Road lane detector function: call the lane detector function to get its lane functions (right and lane points) and the de-warp matrix to adjust it with the input image frames.
- Draw the detected lane: draw both left and right lanes that have been de-warped to the image frames as shown in the Fig. 7 (b).
- Object Position: draw the bounding box of the objects and its positions from the road lane points as shown in the Fig. 7 (c).
- Update the runtime: compute the running time of the proposed method to know its performance in the real-time condition as shown in the Fig. 7 (d).
- Show the detection results: After finishing the update of the runtime, it shows the detection results and the performance of our proposed methods.

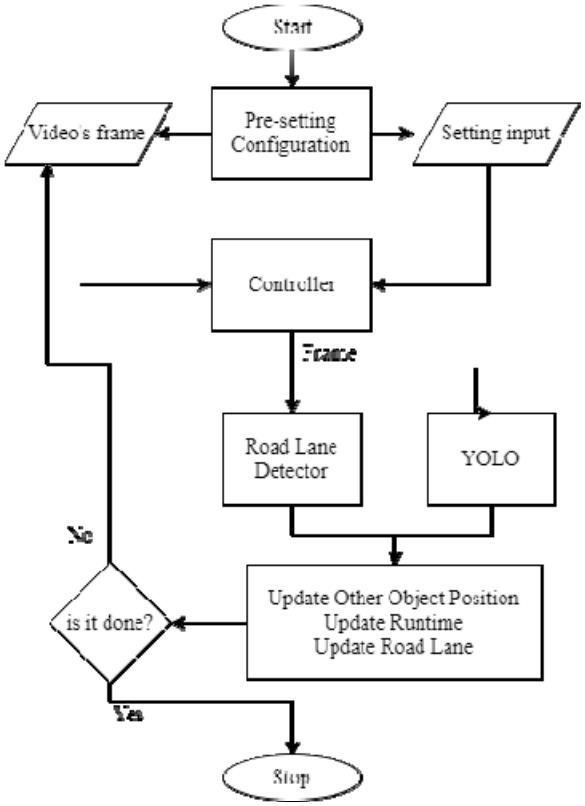

Fig. 5. Controller Flow Process.

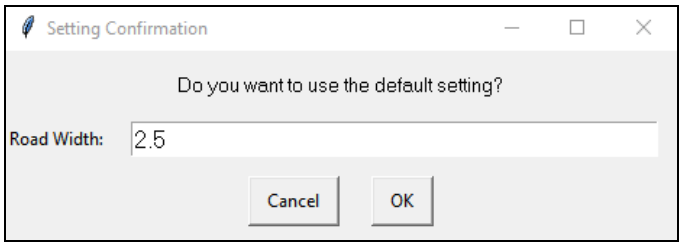

Fig. 6. Pre-setting configuration with default road width value: 2.5.

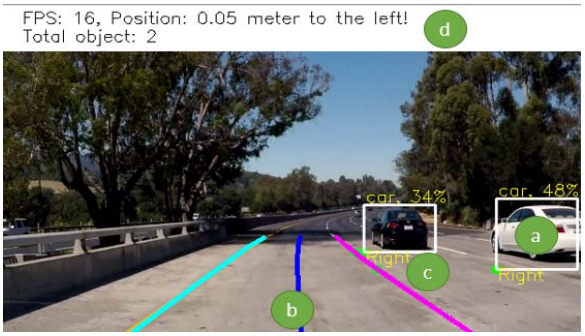

Fig. 7. The Controller functions.

IV. Implementations and Results

A. Implementations

The proposed method's implementations and observations are computed using *Python* programming language and CUDA. For the operating system and other hardware supports, we use Windows 10 Pro, Intel core i5 processor, 16 GB of RAM, and NVIDIA GTX 1070 GPU 8 GB. Table I shows the configuration of the YOLO-small that is used for the implementations with the total of 31 processes.

After the configuration of YOLO has been finished, the controller asks the road width for the pre-setting configuration. Then, it loads the video and gets image frames that will be analyzed by YOLO and road lane detector. Then, it updates the runtime performance, the car position, and the other object positions to give more information to the driver. Finally, it shows all of the information that has been processed on the screen as a real-time feedback for the road lane guidance.

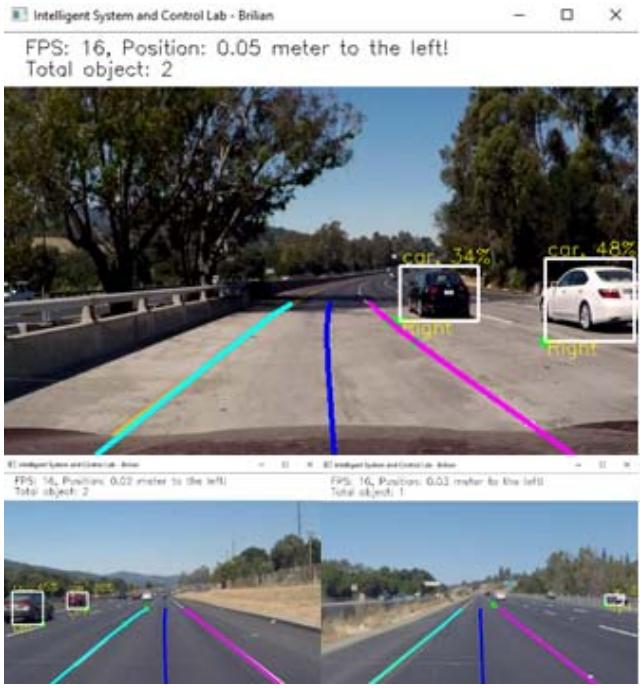

Fig. 8. The results of the detection from the proposed methods on different video recordings in the highway.

B. Results

The results of the implementation are shown in Fig. 8 and Fig. 9 with some screenshots of different videos and the steering direction's suggestions. For the detection suggestion results in the highway areas, it gives pretty accurate results. For the urban area, it seems to be confused between the white line and the sands in the edge of the road.

TABLE I. YOLO-Small Configuration for The Object Detection's Process

Layer 1 : Type = Conv, Size = 7 * 7, Stride = 2, Filters = 64, Input channels = 3
Layer 2 : Type = Pool, Size = 2 * 2, Stride = 2
Layer 3 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 192, Input channels = 64
Layer 4 : Type = Pool, Size = 2 * 2, Stride = 2
Layer 5 : Type = Conv, Size = 1 * 1, Stride = 1, Filters = 128, Input channels = 192
Layer 6 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 256, Input channels = 128
Layer 7 : Type = Conv, Size = 1 * 1, Stride = 1, Filters = 256, Input channels = 256
Layer 8 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 512, Input channels = 256
Layer 9 : Type = Pool, Size = 2 * 2, Stride = 2
Layer 10 : Type = Conv, Size = 1 * 1, Stride = 1, Filters = 256, Input channels = 512
Layer 11 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 512, Input channels = 256
Layer 12 : Type = Conv, Size = 1 * 1, Stride = 1, Filters = 256, Input channels = 512
Layer 13 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 512, Input channels = 256
Layer 14 : Type = Conv, Size = 1 * 1, Stride = 1, Filters = 256, Input channels = 512
Layer 15 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 512, Input channels = 256
Layer 16 : Type = Conv, Size = 1 * 1, Stride = 1, Filters = 256, Input channels = 512
Layer 17 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 512, Input channels = 256
Layer 18 : Type = Conv, Size = 1 * 1, Stride = 1, Filters = 512, Input channels = 512
Layer 19 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 1024, Input channels = 512
Layer 20 : Type = Pool, Size = 2 * 2, Stride = 2
Layer 21 : Type = Conv, Size = 1 * 1, Stride = 1, Filters = 512, Input channels = 1024
Layer 22 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 1024, Input channels = 512
Layer 23 : Type = Conv, Size = 1 * 1, Stride = 1, Filters = 512, Input channels = 1024
Layer 24 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 1024, Input channels = 512
Layer 25 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 1024, Input channels = 1024
Layer 26 : Type = Conv, Size = 3 * 3, Stride = 2, Filters = 1024, Input channels = 1024
Layer 27 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 1024, Input channels = 1024
Layer 28 : Type = Conv, Size = 3 * 3, Stride = 1, Filters = 1024, Input channels = 1024
Layer 29 : Type = Full, Hidden = 512, Input dimension = 50176, Flat = 1, Activation = 1
Layer 30 : Type = Full, Hidden = 4096, Input dimension = 512, Flat = 0, Activation = 1
Layer 32 : Type = Full, Hidden = 1470, Input dimension = 4096, Flat = 0, Activation = 0
INFO:tensorflow:Restoring parameters from weights/YOLO_small.ckpt

Fig. 9. The results of the detection from the proposed methods on different video recordings in the highway.

V. Discussions and Limitations

In the road lane detector, we did some filtering and warping before we detect the lane points. The image filtering limits us to accommodate the undetected road lanes and restraints us from a robust detection system. Besides that, sudden over lighting/illumination to recording camera/environment still greatly affects the detection of our methods even if we do normalizations, i.e. Contrast Limited Adaptive Histogram Equalization (CLAHE) and gamma correction. The road lane detection from over lighting is shown in the Fig. 10.

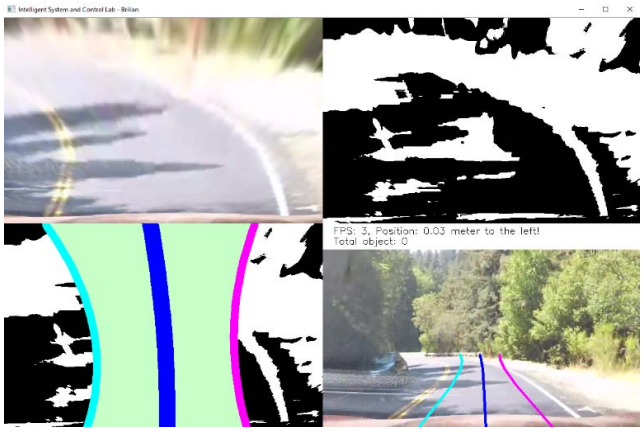

Fig. 10. The over lighting effects to the camera (above) and the urban road environment (below).

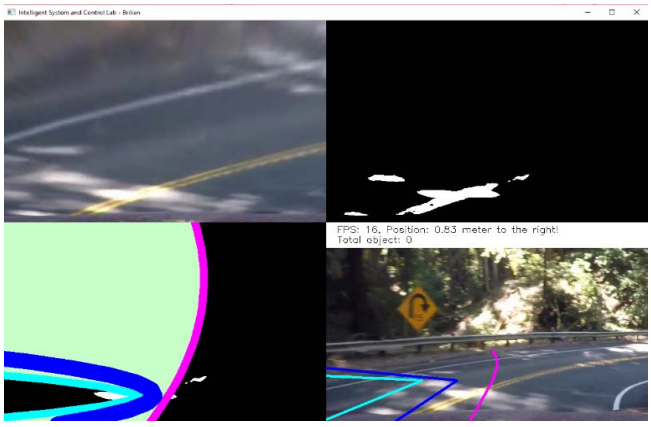

Fig. 11. Results of the system of the road lane detector on a sharp-turned road on urban street.

The road lane detector also uses the crop of left and right to find the lane points in both lanes. While the approach is pretty good for straight and non-sharp-turned roads. However, it brings out a limitation, where the sharp-turned roads could lead to inaccurate detection as shown in the Fig. 11.

VI. Conclusions and Futureworks

We have demonstrated a detail and step-by-step approach for the road lane guidance. The proposed method use pre-trained YOLOv1, road lane detector, and a controller to integrate them into a smart system. Based on the observations, the proposed methods can adjust the road lane, detect the objects (e.g. cars), and provide the steering suggestion to the driver to help the automation of the driving process.

This paper is suitable for highway conditions, when it mostly fails to detect the road lane in the urban roads. With those limitations in mind, we will try to make better and robust system that can handle over lighting, sharp-turned roads, and collision warning, and different road environments by using end-to-end learning using CNN to increase the capability of our system to be applied in the real environment of self-driving cars.

Acknowledgment

The resources of implementations for this experiment are provided by The Intelligent System and Control (ISC) lab of NTUST.

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1-9, 2015.

[2] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: towards realtime object detection with region proposal networks,” Advances in Neural Information Processing Systems (NIPS), 2015.

[3] R. Girshick, “Fast r-cnn,” Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 1440-1448, 2015.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, “SSD: single shot multibox detector,” Proceedings of the European Conference on Computer Vision (ECCV), 2016.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: unified, real-time object detection,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779-788, 2016.

[6] J. Redmon and A. Farhadi.”YOLO9000: Better, Faster, Stronger,” IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6517-6525, 2017.

[7] N. Kanopoulos, N. Vasanthavada, and R.L. Baker, “Design of image edge detection filter using the sobel operator,” IEEE Journal Of Solid-State Circuits, vol. 23, no. 2, 1988.

[8] G.T. Shrivakshan, “A comparison of various edge detection techniques used in image processing,” IJCSI International Journal of Computer Science (IJCS), vol. 9, no. 3, pp. 358-367, 2012.

[9] J. Canny, “A computational approach to edge detection,” IEEE Transactions On Pattern Analysis And Machine Intelligence (PAMI), vol. 8, no. 6, pp. 679-698, 1986.

[10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet large scale visual recognition challenge,” International Journal of Computer Vision (IJCV), vol. 115, pp. 211-252, 2015.