Design of ADAS for Vehicles using Single-Shot Multibox Detector (SSD)

[1]Sudharshan S M, [2]Dr. Kiran A Gupta

[1]Student, MTech, VLSI Design and Embedded Systems
[2]Professor, Dept. of Electronics and Communication Engineering
[1,2]Dept. of Electronics and Communication Engineering,
[1,2]Dayananda Sagar College of Engineering, Bengaluru, Karnataka, India

Abstract : Advanced Driver Assistance System (ADAS) is the most researched area in the automobile industry since the last decade. ADAS includes different features that help the driver for a safer and easier driving. In this work, we have designed an ADAS system that incorporates camera and ultrasonic sensors for object detection in real-time. The system has Raspberry Pi 3 model B+ for processing. Single Shot multibox Detector (SSD) algorithm is used for real-time object detection in the video. We have achieved an average precision of about 74.3% at an average of 35fps (frames per second). The ultrasonic sensor (HC-SR04) used at the rear of the vehicle has a precision of 0.1-0.5cm and range between 2cm to 400cm**.**

***IndexTerms* - Deep learning, SSD, YOLO, real-time video processing, ADAS.**

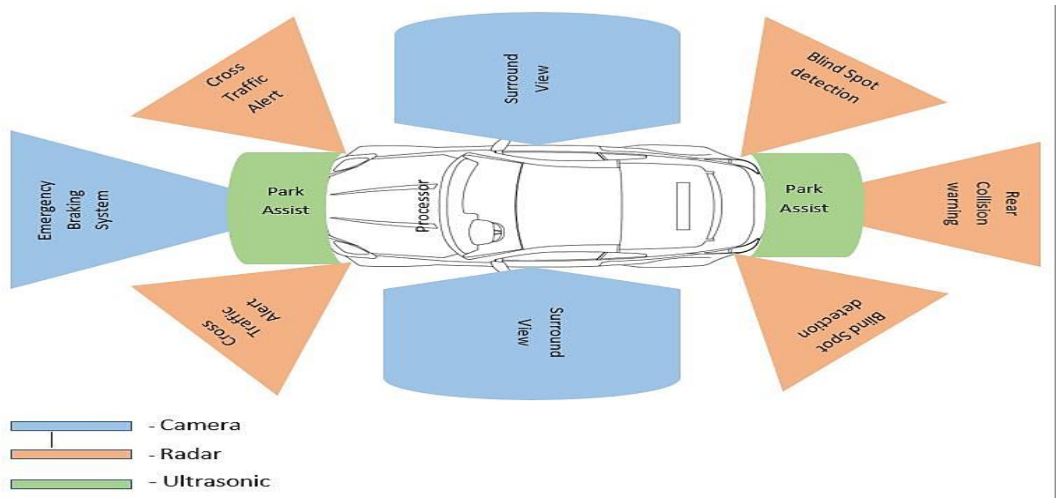
I. INTRODUCTION

Each year more than 1,50,000 people are killed in road accidents in India alone[1]. That is about 400 deaths in a day. Accidents are happening due to variety of reasons: speeding two-wheelers, cars not equipped with airbags, breaking of traffic rules, poor maintenance of roads etc. This has been a major concern for the government. New rules and regulations have been introduced in this context and the automobile industries have been ordered to include certain safety features in their vehicles. This paved the way for Advanced Driver Assistance System (ADAS) [8] [9]. ADAS can be of any complexity. i.e, different features can be embedded in it. Some of the common features include parking assistance, navigation, vehicle detection in the same lane, assistance during lane changes, traffic sign detection and pedestrian detection. General features that can be included in an ADAS for a car is shown in Fig 1.

Advancements in Artificial Intelligence (AI) and new methodologies in deep learning is playing a major role in the design and development of ADAS. Majority of the methods that are currently available are unable to meet the requirements exhibited by the automobile industries today. These methods are limited by slow processing time. Hence, this raised a requirement for a method that has better performance time while maintaining the accuracy. Convolutional Neural Networks (CNN) were introduced in the recent years which transformed the way of processing real-time signals. Accuracy of CNN-based methods are much better compared to that pre-existing methods [2]. However, it is very complex and time-consuming. Hence, certain steps were needed to optimize these methods so that they can be used for real-time application. CNN-based detectors are usually divided into two categories; two-stage and single stage. RCNN family (RCNN, Fast RCNN, and Faster RCNN), which is a two-stage algorithm, have a much better accuracy than other detection methods [3]. However, these methods require more processing time as their computational cost is higher. You Only Look Once (YOLO) [4] and Single shot multi-box detector (SSD) are single-stage detectors. Single stage detectors are faster and have less computational cost than two stage detectors. This is because single stage detectors detect the object, localize it, determine its class label and confidence of detection in single forward pass on the input image. They are not as accurate as two stage detectors but are faster which is a trade-off.

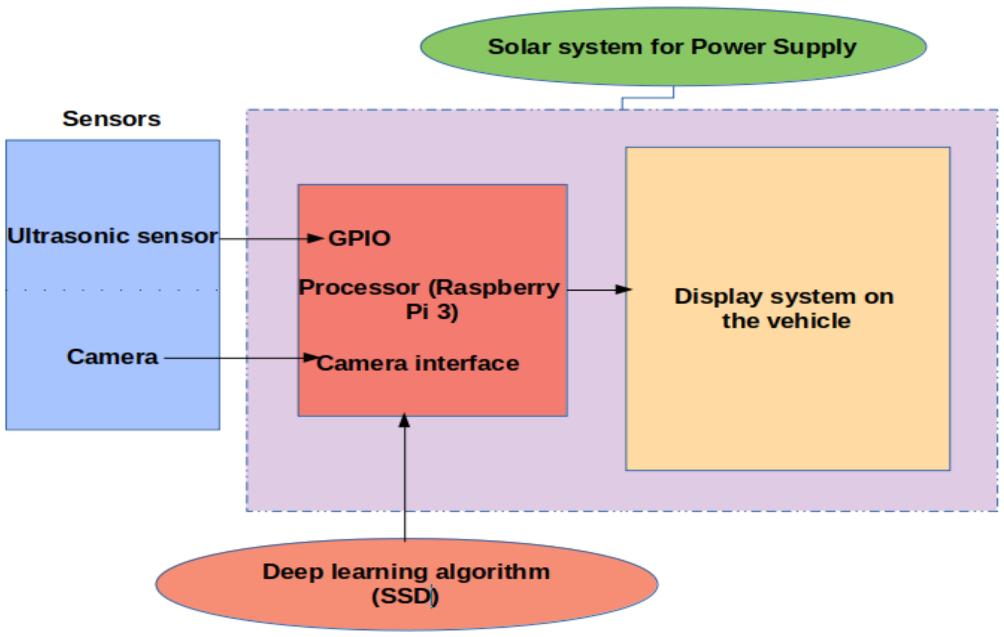
YOLO can detect objects faster than RCNN family. YOLO considers object detection as a single regression problem. It takes the input image, predicts multiple bounding boxes and their class probabilities. YOLO divides the input image into an SxS grid and bounding boxes, confidence for these boxes and class probabilities are predicted. But it fails when the object size is small or if multiple objects are grouped closely. This is because the YOLO algorithm a single object in each of the cells of SxS grid. If there are multiple objects or very small objects in the cell, they will be missed.

Single Shot multibox Detectors (SSD) [5] have achieved a balance between faster processing and better detection than YOLO. In this paper, we have used SSD algorithm for object detection. In the coming sections we discuss about SSD and its basic architecture, methodology of proposed ADAS and the obtained results.



II. PROPOSED METHOD

The block diagram of the proposed system is shown in Fig 2. Camera and ultrasonic sensors are used for taking raw data. Camera is placed on the car's dashboard such that the front of the vehicle is captured. Detection of vehicles and pedestrians in front of the vehicle can be done using this. Raspberry Pi camera is used for this purpose in the proposed system. The camera is 5MP and can record video at different resolutions and speed such as 1080p @30fps, 720p @60fps and 640x480p @60/90fps. Video of any resolution can be processed by the system developed.


Solar system for Power Supply
Sensors
Ultrasonic sensor
Camera
GPIO
Processor (Raspberry Pi 3)
Camera interface
Display system on the vehicle
Deep learning algorithm (SSD)

Along with the camera, ultrasonic sensor is used to monitor any objects at the rear of the vehicle. The ultrasonic sensor used is HC-SR04. It has an accuracy of ~0.1cm and a range of 2cm to 400cm. It has 4 pins: i) VCC, ii) GND, iii) Trig and iv) Echo. The sensor is powered using the GPIO pins of the Raspberry Pi. 10 µs pulses are generated at the trigger pin and the echo pin is monitored. The distance is calculated using the speed of the sound in the air and the time taken by the signal to bounce back from the object and reach echo pin. The simplified equation is as shown in (1).

$$Distance\ = Time\ x\ 17150 \qquad (1)$$

Sensors are interfaced with Raspberry Pi 3 model B+. This has Broadcom BCM2837B0 quad-core A53 processor built on ARMv8 architecture, 64bit and has a clock speed of 1.4GHz. 1GB RAM and 40 GPIO pins are available on this model.

2.1 Single Shot Multibox Detectors

The basic architecture of Single Shot multibox Detectors (SSD) is shown in Fig 3[5]. The input image is of 300x300 resolution. VGG16 is the basic feature extraction network in SSD and it is followed by other convolutional layers that help in obtaining more feature maps. The image is passed through all these layers producing feature maps of different sizes which are then evaluated to generate bounding boxes. Detection of smaller objects are done in the early phase where the image resolution is high and detection of larger objects are done accurately in the later stages. Multiple bounding boxes will be generated for the detected objects. SSD uses priors for accurate bounding box generation. Priors are created such that their dimensions are close to ground truth boxes and then intersection over union (IoU) of generated boxes and priors is calculated to determine the suitable bounding box for the detection. IoU value of the bounding box should be more than a threshold, say 0.5. Only then that particular box of all the multiple boxes generated for locating the object is considered as bounding box. Higher the IoU better is the bounding box accuracy.

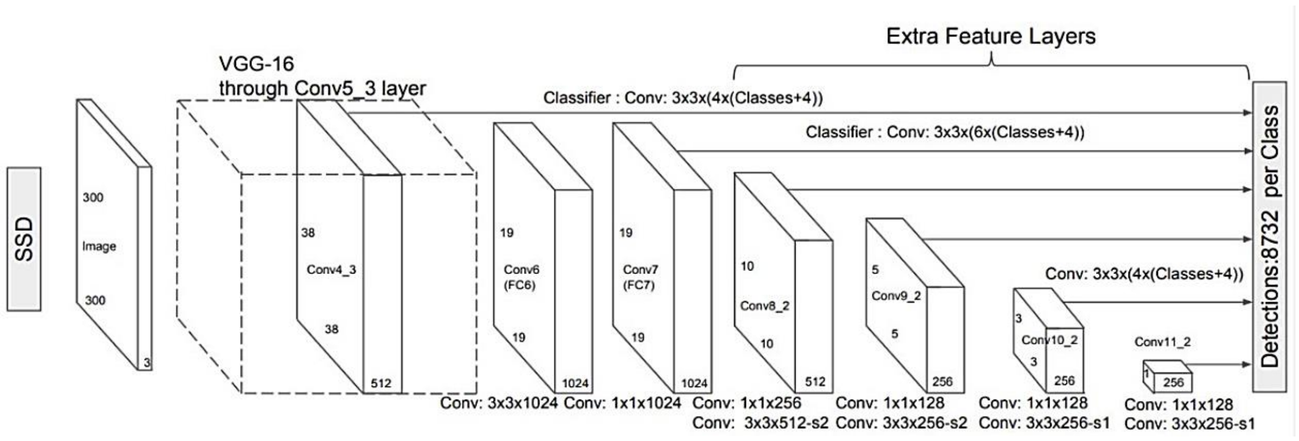


Fig SEQ Figure * ARABIC 2: Block diagram of the proposed system

2.2 Real-time object detection

Proposed object detection method is shown using the flowchart in Fig 4. Multiprocessing is used to improve the speed of detection. The main process, shown in Fig 5, initializes the camera and starts the fps counter. Also, the pretrained model is loaded. An input queue, to store the incoming video frames, and an output queue, to place the detection, are initialized. The frames recorded are sent to input queue and detection happens in the child process as shown in Fig 6. Frames in the input queue are preprocessed using the deep neural network function available in OpenCV library. The frame will be resized to 300x300 and scaled. Also, a mean value of 127.5 is used to neutralize illumination abnormalities. Once the preprocessing is done, a blob is created and it is sent to object detector. The object detector is trained using Microsoft's COCO (common objects in context) data set [6]. Then it is fine-tuned using MobileNets [7]. MobileNets helps in building light weight deep neural networks that uses Rectified Linear Unit (ReLU) as activation function. Using MobileNets helps in model size reduction and also increases processing speed. Once the detections are done, the results are put into the output queue. The child process then takes the next frame from the input queue and the steps are repeated.

Fig SEQ Figure * ARABIC 3: Architecture of SSD

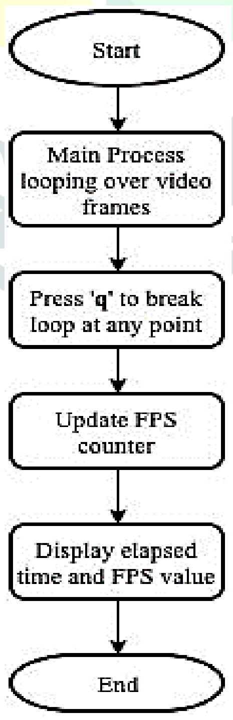

Start
Main Process looping over video frames
Press 'q' to break loop at any point
Update FPS counter
Display elapsed time and FPS value
End

Fig 4: Flowchart of the entire object detection process

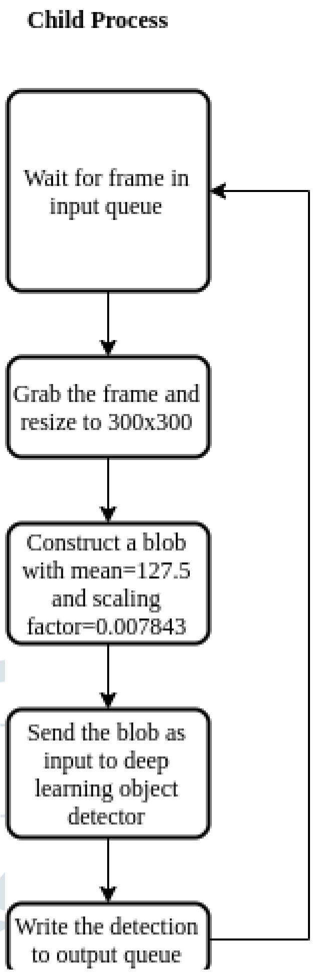

Child Process
Wait for frame in input queue
Grab the frame and resize to 300x300
Construct a blob with mean=127.5 and scaling factor=0.007843
Send the blob as input to deep learning object detector
Write the detection to output queue

Fig 6: Flowchart of child process

In the main process, the output queue is checked for any detections. If there are some detections then their confidence is evaluated against the threshold value set at the beginning. If the confidence is less than the threshold then that detection is discarded. Otherwise the class label of the detection is obtained and the (x,y) coordinates are computed. Draw these predictions on the original frame and display the class label and confidence of detection along with the bounding box representing the object location. Then check if the detected object is too near to your vehicle by monitoring its x and y coordinates in the output frame. Set threshold values for x and y coordinates such that they define the size of nearby objects. If the x and y values are more than threshold then display a warning message on the screen to alert the driver. The entire loop in the program can be stopped at any point by pressing a key (here, we have taken ’q’). Once the loop is broken update the fps counter and display the elapsed time and fps value.

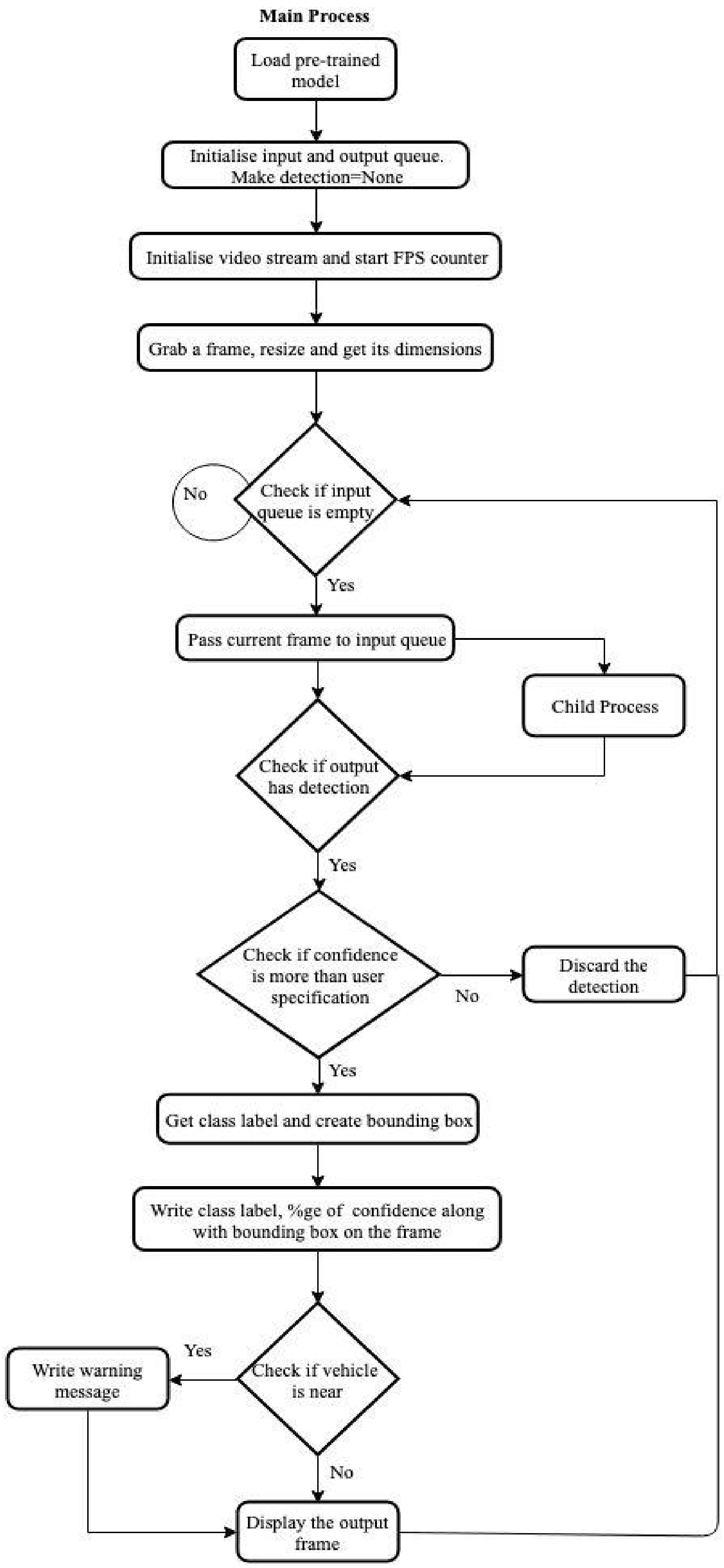
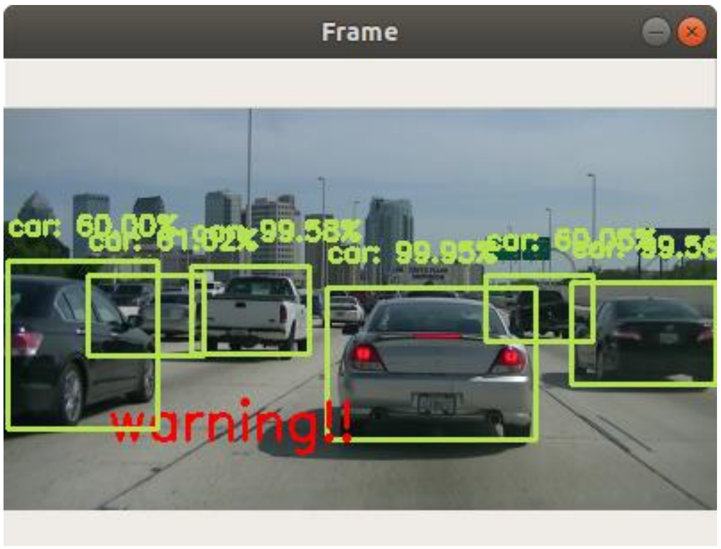
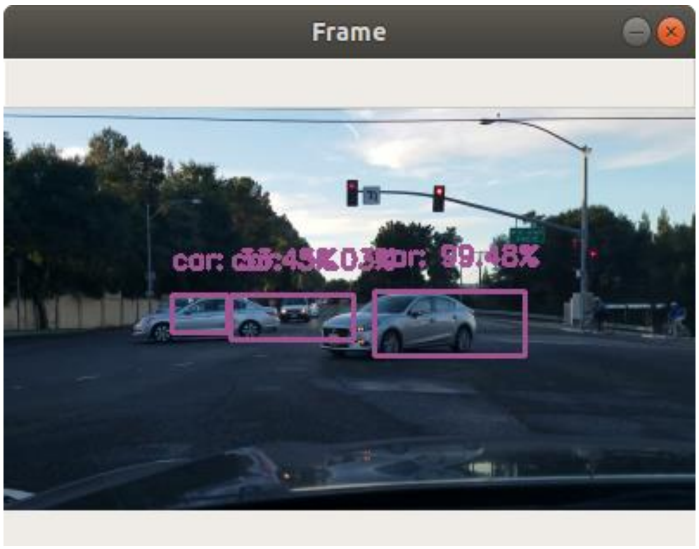

Main Process
Load pre-trained model
Initialise input and output queue. Make detection=None
Initialise video stream and start FPS counter
Grab a frame, resize and get its dimensions
No
Check if input queue is empty
Yes
Pass current frame to input queue
Child Process
Check if output has detection
Yes
Check if confidence is more than user specification
No
Discard the detection
Yes
Get class label and create bounding box
Write class label, %ge of confidence along with bounding box on the frame
Yes
Write warning message
Check if vehicle is near
No
Display the output frame

Fig 5: Flowchart of main process

III. RESULTS





Output is shown for different conditions: i) Detection of vehicles in front, on a highway. Warning is displayed when a vehicle is too close as in Fig7. ii) Detection of vehicles at an intersection, Fig8. iii) Detection of both vehicles and multiple pedestrians is displayed Fig 9. Even the smaller objects are detected successfully.

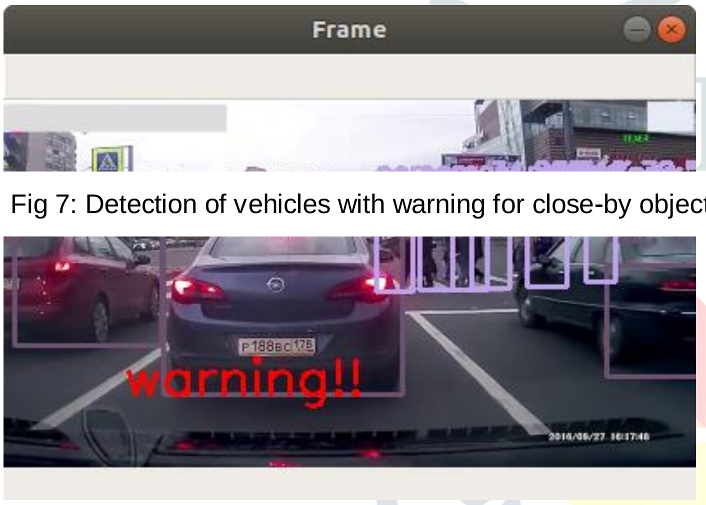


Fig 7: Detection of vehicles with warning for close-by object

Fig 9: Detection of vehicle and pedestrians

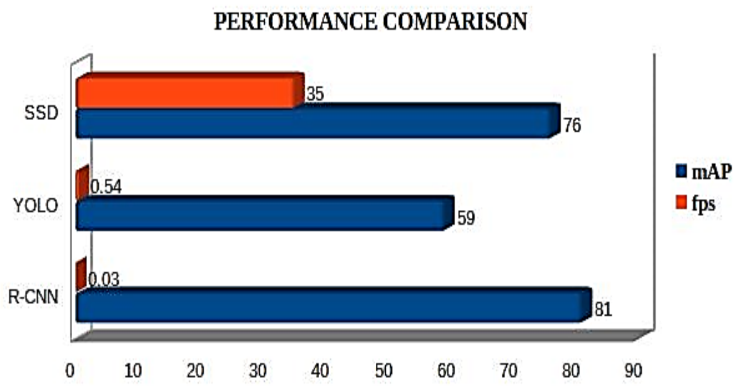


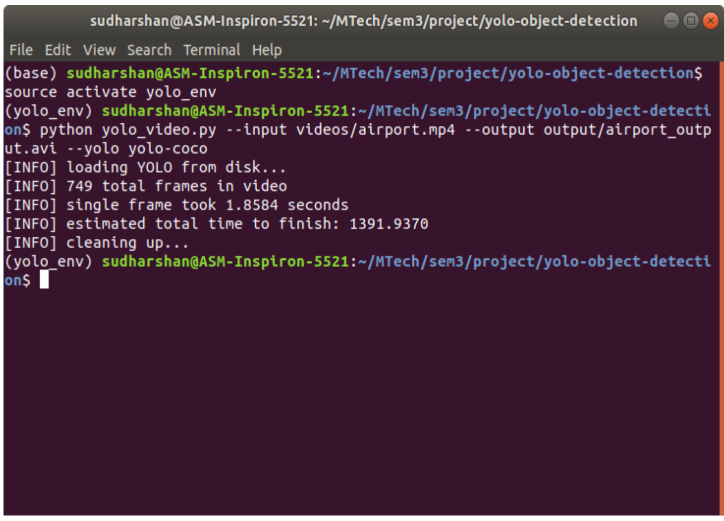
Fig 10: Performance comparison of different algorithms

3.1 Comparison

The figures 11,12 below show the frame rate of both YOLO and SSD algorithms for the same video input. Also, Fig13 shows the output of the proposed system on the Raspberry Pi 3 Model B+, with both camera and ultrasonic sensor running simultaneously. As you can see, SSD is much faster than YOLO. SSD implemented in this paper has the input image of size 300x300 and that of YOLO is 416x416. Detailed comparison is discussed in table I. Faster R-CNN takes an input image of resolution 1000x600 [3]. All the methods have been trained on COCO dataset [6].

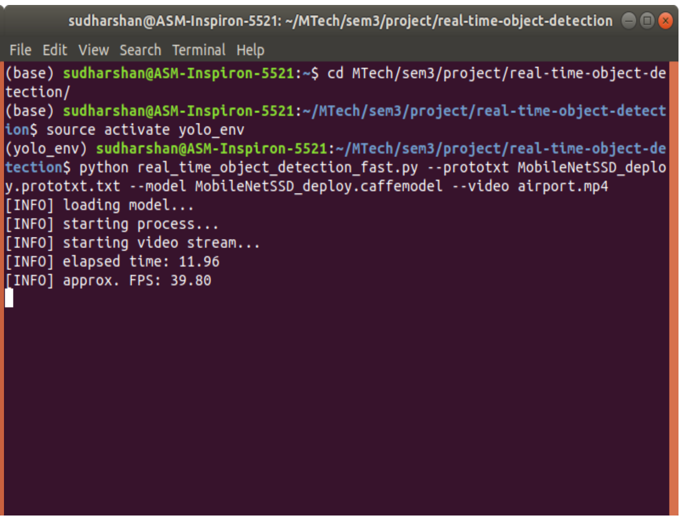
Algorithm	Dataset	Fps	mAP	Input Resolution
(Faster)R-CNN [3]	COCO	0.03	73.2	~1000x600
YOLO	COCO	0.54	66.4	416x416
SSD	COCO+MobileNets	35	74.3	300x300

Table 3.1: Comparison of different algorithms.


sudharshan@ASM-Inspiron-5521: ~/MTech/sem3/project/yolo-object-detection
File Edit View Search Terminal Help

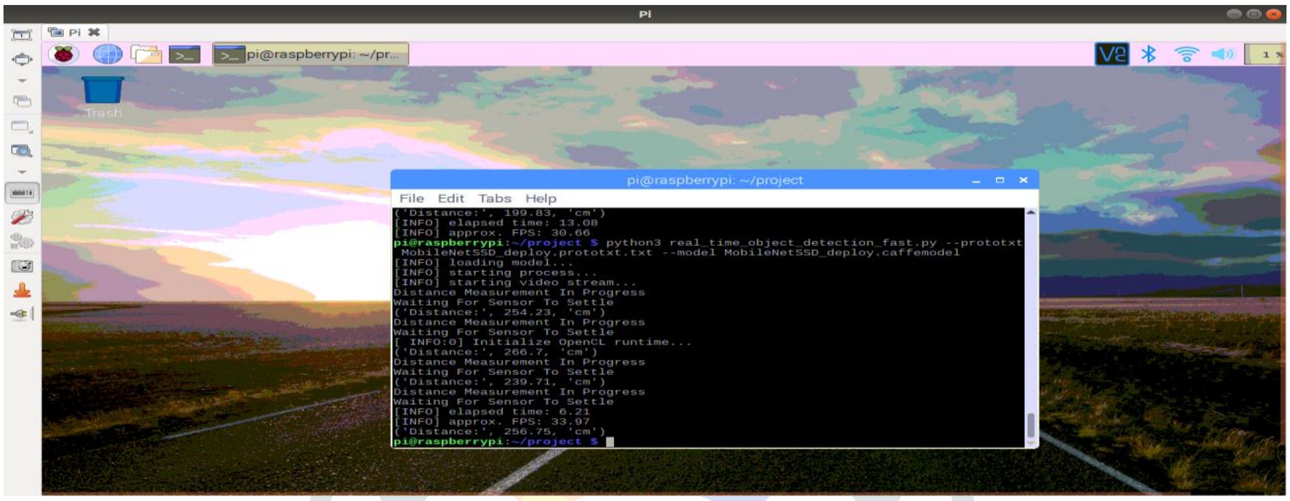
```
(base) sudharshan@ASM-Inspiron-5521:~/MTech/sem3/project/yolo-object-detection$
source activate yolo_env
(yolo_env) sudharshan@ASM-Inspiron-5521:~/MTech/sem3/project/yolo-object-detecti
on$ python yolo_video.py --input videos/airport.mp4 --output output/airport_outp
ut.avi --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] 749 total frames in video
[INFO] single frame took 1.8584 seconds
[INFO] estimated total time to finish: 1391.9370
[INFO] cleaning up...
(yolo_env) sudharshan@ASM-Inspiron-5521:~/MTech/sem3/project/yolo-object-detecti
on$
```

Fig 11: Frame rate in YOLO algorithm


sudharshan@ASM-Inspiron-5521: ~/MTech/sem3/project/real-time-object-detection
File Edit View Search Terminal Help

```
(base) sudharshan@ASM-Inspiron-5521:~$ cd MTech/sem3/project/real-time-object-de
tection/
(base) sudharshan@ASM-Inspiron-5521:~/MTech/sem3/project/real-time-object-detect
ion$ source activate yolo_env
(yolo_env) sudharshan@ASM-Inspiron-5521:~/MTech/sem3/project/real-time-object-de
tection$ python real_time_object_detection_fast.py --prototxt MobileNetSSD_deplo
y.prototxt.txt --model MobileNetSSD_deploy.caffemodel --video airport.mp4
[INFO] loading model...
[INFO] starting process...
[INFO] starting video stream...
[INFO] elapsed time: 11.96
[INFO] approx. FPS: 39.80
```

Fig 12: Frame rate in SSD algorithm



```
('Distance:', 199.83, 'cm')
[INFO] elapsed time: 13.08
[INFO] approx. FPS: 30.66
pi@raspberrypi:~/project $ python3 real_time_object_detection_fast.py --prototxt
 MobileNetSSD_deploy.prototxt.txt --model MobileNetSSD_deploy.caffemodel
[INFO] loading model...
[INFO] starting process...
[INFO] starting video stream...
Distance Measurement In Progress
Waiting For Sensor To Settle
('Distance:', 254.23, 'cm')
Distance Measurement In Progress
Waiting For Sensor To Settle
[ INFO:0] Initialize OpenCL runtime...
('Distance:', 266.7, 'cm')
Distance Measurement In Progress
Waiting For Sensor To Settle
('Distance:', 239.71, 'cm')
Distance Measurement In Progress
Waiting For Sensor To Settle
[INFO] elapsed time: 6.21
[INFO] approx. FPS: 33.97
('Distance:', 256.75, 'cm')
pi@raspberrypi:~/project $
```

Fig 13: Output on Raspberry Pi 3 Model B+

CONCLUSION

The proposed ADAS system can detect objects in real-time with a good mean average precision of about 74.3% at around 35fps. It even detects smaller objects better than YOLO and is faster as shown in results. Single Shot multibox Detectors (SSD) are trained on COCO dataset and then fine-tuned using MobileNet. The algorithm is tested on both computer and Raspberry Pi 3 model B+ and the above mentioned performance is obtained. The ultrasonic sensor at the rear of the vehicle also detects objects that are up to 400cm away with high precision. Hence, the proposed system outperforms other deep learning algorithms in terms of processing speed in real-time.

REFERENCES

[1] Miss. Indrayani B. Dhotre, et al., “Advance Driver Assistance System for Indian Traffic Scenario: Literature Review”, COMPUSOFT, An international journal of advanced computer technology, 6 (1), January- 2017 (Volume-VI, Issue-I).
[2] Hai Wang, et al., “A Comparative Study of State-of-the-Art Deep Learning Algorithms for Vehicle Detection”, IEEE Intelligent transportation systems magazine, 2015.
[3] Yongjie Zhang et al., “Real-time vehicle detection and tracking in video based on faster R-CNN”, J. Phys.: Conf. Ser. 887 012068, 2017.
[4] Joseph Redmon, et al., “You Only Look Once: Unified, Real-Time Object Detection”, arXiv:1506.02640v5 [cs.CV], 9 May, 2016.
[5] Wei Liu, et al., “SSD: Single Shot MultiBox Detector”, arXiv:1512.02325v5 [cs.CV], 29 Dec,2016.
[6] Tsung-Yi Lin, et al., “Microsoft COCO: Common Objects in Context”, arXiv:1405.0312v3 [cs.CV], 21 Feb, 2015.
[7] Andrew G. Howard, et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, arXiv:1704.04861v1 [cs.CV], 17 Apr, 2017.
[8] Abdallah Moujahid, et al., “Machine Learning Techniques in ADAS: A Review”, International Conference on Advances in Computing and Communication Engineering (ICACCE-2018) Paris, France, June 2018.
[9] Irfan Baftiu, et al., “Multi-Mode Surround View for ADAS Vehicles”, IEEE International Symposium on Robotics and Intelligent Sensors (IRIS2016), Tokyo Japan, December 2016.