ISSN No: 2456-2165

Shallow Convolution Neural Network for an Industrial Robot Real Time Vision System

Dumindu Eranda Jayakody,
Department of Mechanical Engineering,
The Open University of Sri Lanka, Sri Lanka.

J.A.K.S. Jayasinghe,
Prof., Department of Electronics and Telecommunication Engineering,
Faculty of Engineering,
University of Moratuwa, Sri Lanka.

D. C. Wijewardene,
Eng., Department of Mechanical Engineering,
The Open University of Sri Lanka, Sri Lanka.

Abstract:- Recent advancement in the deep learning-based object detection techniques have significantly improved the detection speed and accuracy. Now, it is possible to execute a complex deep neural network having a large number of layers in real time using desktop computers equipped with graphic processing units (GPU's). Further, shallow neural network architectures are also developed for embedded systems with less processing power. This paper proposes shallow neural network architecture capable of real-time object detection as a vision system of an industrial robot. The proposed shallow neural network is executed on Raspberry Pi 3B+ having limited resources compared to a desktop computer equipped with GPU's.

Keywords:- *Computer Vision, Object Detetion, Yolo.*

I. INTRODUCTION

Computer vision is a branch of computer science, which enables machines not only to see but also to process and analyze images and videos. One of the key areas of computer vision is object detection. It gives capability to computers to locate and identify objects in an image. Application of Deep Learning and Convolution Neural Network (CNN) techniques to computer vision has created an important milestone [1]. Using these techniques, computers can learn from large training image data sets to recognize and track objects.

A vision system capable of real-time object detection and tracking plays a crucial role in an intelligent robot. Such a robot can interact with the environment using visual data captured by a camera. With a real-time vision system [2][3], robot can execute tasks automatically by tracking objects while estimating orientation and velocity of the objects. With such features, vision system can guide the robot manipulator to interact with the object within a very short time period. Further, robot will be able to manage unpredictable events such as a human coming close to the danger zone of the robot.

In this paper, we present implementation of a real-time vision system using a resource limited Single Board Computer (SBC) having a very small footprint. The small footprint of the SBC makes it feasible to incorporate with an industrial robot.

II. SYSTEM ARCHITECTURE

There are three subsystems in our architecture as shown in Fig.1. Camera captures the image and Image Processor implements the object detection while the Robot Controller controls the robot based on the information received from the Image Processor. The Image Processor carries out the object detection and sends the location and orientation of the detected object to the Robot Controller. Logitech C270 web camera is used as the camera and Raspberry Pi 3B+ is used as Image Processor in our system.

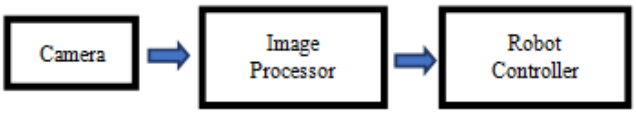

Figure 1: System Block Diagram

Following two methods were considered for improving the processing speed of the Image Processor implemented on the selected SBC:

- Customized YOLO Algorithm based on YOLO-Lite [3]
- Bare-metal Programming [4]

Bare-metal Programming

CNNs are usually programmed using scripting languages like Python, which have more support libraries and more community forums. In such implementations, the CNN can be slowed down due to operating system overheads. If the program is executed without an operating system, performance can be improved. Bare-metal programming is used in such cases. Bare-metal programming interacts with the processor system at the hardware level without the underling layers of the operating system. Unfortunately, the necessary datasheets of the new powerful versions of

ISSN No: 2456-2165

Raspberry Pi (such as 3B+) are not available to implement the CNN using the bear-metal programming method.

Customized Yolo Algorithm based on Yolo-Lite

The architecture of a Yolo algorithm is specified by the configuration file (.cfg file) [2],[3],[5],[6]. It contains information such as number of convolution layers, number of filters in each layer, type of activation function used in each layer and how image size is reduced from a given convolution layer to next convolution layer. This file is a human-readable file and can be customized to implement modified CNN algorithms.

Following fields in the Yolo configuration file are considered for customization:

- Input layer size
- Batch normalization
- Number of Convolutions layers
- Lowering the number of objects to be identified

1) Input layer size

The Yolo-Tiny [6] is a well-known algorithm which processes a 416*416 input image. As the calculations are done on floating-point numbers, which is slow compared to fixed-point or integer computations, the speed of the convolution neural network heavily depend on the input image size. The amount of calculation can be reduced by using a low-resolution image which tends to increase the error rate.

2) Batch Normalization

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and reducing the number of training epochs required to train deep networks.

3) Number of Convolution layers

Convolution layers are the major building blocks used in convolution neural networks. Convolution is the simple application of a filter to an input image. The convolution neural networks have the ability to automatically tune a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modeling, such as image classification.

Amount of calculation required for processing can be reduced by using a few convention layers which again tends to increase the error rate.

4) Lowering the number of objects

Yolo-Tiny[6] algorithm can identify 20 objects when it use PASCAL VOC data set. This number is decided by the size of the last layer of the convolutional neural network. Amount of computations required for the convolutional neural network can also be reduced by lowering the number of detectable objects.

Based on these facts, we have studies three different CNN architectures with following basic features summarized in Table1.

Table 1 : Customized Yolo Algorithms Summarize

Architecture	Input layer size	Batch Normalization	Number of Convolution layers	Stride
1	224*224	yes	8	2
2	224*224	yes	7	2
3	128*128	yes	7	2

In [5], there are 21 different trials and they have tested them using Dell XPS 13 which is a non-GPU based laptop. Their target is to have 10 frames per second (fps) object detection rate. Out of 21 different trials, 6 trials have shown fps higher than 10. As the Raspberry Pi has less computational power, we selected two trails from [5] having the lowest floating-point operations per second (flops) and highest fps rate (i.e., trial 12 & trail 3).

In order to select the best architecture for the Raspberry PI, we decided to train them using two objects (Hand and TV antenna cable end). Then the selected architecture was trained for more objects.

III. RESULTS

We use Google Colaboratory (Google Colab) to train the architectures. Then all architectures we tested on the Raspberry Pi 3B+ to measure the frame rate (fps). With the configuration file, one can enable or disable batch normalization option. As Google Colab platform does not support batch normalization disabled architectures, we have studied all the architectures with batch normalization.

For selection of the best architecture following measures were considered.

- Fps
- Flops
- mAP (Mean average precession)

mAP is a measure about the network accuracy.

1) Architecture 1

Due to the low flops count, Yolo-Lite [5] trail 12 was selected as Architecture 1 and tested on Raspberry Pi 3B+. Results are shown in Fig. 2 and Fig. 3 where the percentage number shows the confidence of the detected object. Key features of Architecture 1 are as follows:

- flops count – 71M
- mAP value – 26.90%
- fps on Dell XPS 13 – 6.9
- fps on Raspberry Pi 3B+ – 2.9

ISSN No: 2456-2165

Figure 2 : Architecture 1 Result for TV Antenna Cable End Detection on Raspberry Pi 3B+

Figure 3: Architecture 1 Result for Hand Detection on Raspberry Pi 3B+

2) Architecture 2

Due to the moderate flop requirement and high mAP, trail 3 of YOLO-Lite [5] was selected and tested on Raspberry Pi 3B+. Results are shown in Fig. 4, Fig. 5 and key measures are as follows:

- flops count – 482M
- mAP value – 34.59%
- fps on Dell XPS 13 – 9.5
- fps on Raspberry Pi – 4.9

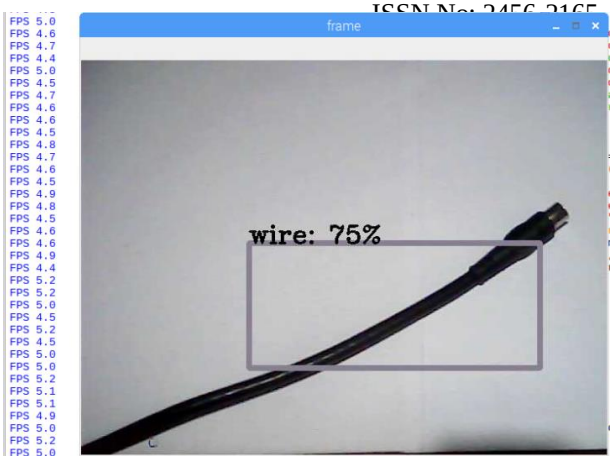

Figure 4: Architecture 2 Result for TV Antenna Cable End Detection on Raspberry Pi 3B+

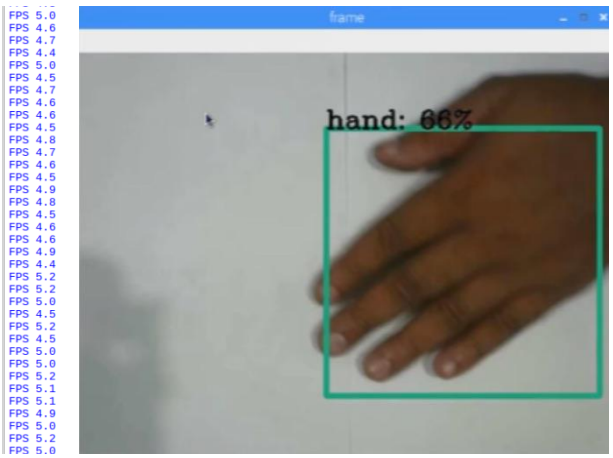

Figure 5: Architecture 2 Result for Hand Detection on Raspberry Pi 3B+

3) Proposed Architecture

Due to low processing power on the Raspberry Pi 3B+, we noted that it is not possible to achieve 10 fps detection rate. Hence, we study several architectures and we found that proposed Architecture as detailed in Table 2, can achieve 10 fps detection rate.

Table 2 : Proposed Architecture

Layer	Filters	Size	Stride	Pad	Number of pixels
Input image (128, 128)	-	-	-	-	-
Convolution 1	16	3×3	1	1	128, 128,16
MaxPooling	16	2×2	2	-	64, 64
Convolution 2	32	3×3	1	1	64, 64, 32
MaxPooling	32	2×2	2	-	32, 32
Convolution 3	64	3×3	1	1	32, 32, 64
MaxPooling	64	2×2	2	-	16, 16
Convolution 4	128	3×3	1	1	16, 16, 128
MaxPooling	128	2×2	2	-	8, 8
Convolution 5	128	3×3	1	1	8, 8, 128
MaxPooling	128	2×2	2	-	4, 4
Convolution 6	256	3×3	1	1	4, 4, 256
MaxPooling	256	2×2	2	-	4, 4
Convolution 7	35	1×1	1	1	4, 4, 35

Performance of our proposed architecture on Raspberry Pi 3B+is shown in Fig. 6, Fig. 7 and key measures are as follows:

- flops count – 60M
- mAP value - 30.15%
- fps rate in Raspberry Pi 3B+- 10.5

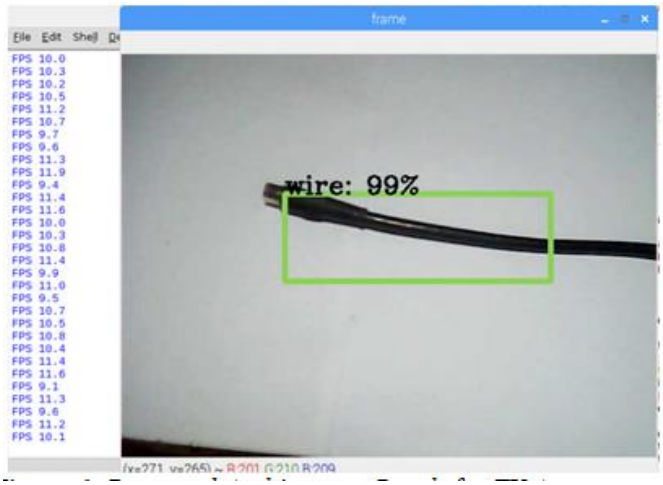

Figure 6: Proposed Architecture Result for TV Antenna Cable End Detection on Raspberry Pi 3B+

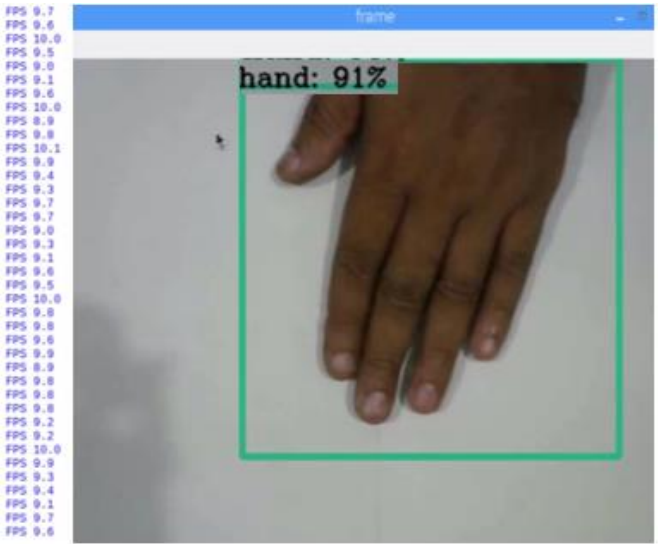

Figure 7: Proposed Architecture Result for Hand Detection on Raspberry Pi 3B+

Table 3 : Architecture Comparison

Architecture	Flops count	mAP value	Fps rate on Raspberry Pi	Fps rate on Dell XPS 13
1	71M	26.90%	2.9	6.9
2	482M	34.59%	4.9	9.5
Proposed	60M	30.15%	10.5	-

In order to test the suitability of the proposed architecture for applications in intelligent robots, we considered the following event. There are three types of milk packets in a tray such that the intelligent robot detects the type of milk packet and pick it and place in three different bins. Due to safety reasons, the intelligent robot will carry out the operation if no human hand is visible. In this case, five different objects (milk packet type1, milk packet type2, milk packet type3, TV antenna cable end and Hand) were taken for training. Fig. 8 to Fig. 12 show the performance and we observed the frame rate is marginally dropped to 9.5. The reason for dropping the FPS is to increase the objects for the final layer filter and add the center and direction of the object for calculations. Further, center and direction of the object also detected (as show by red marker and blue line) after the objects were detected by the CNN algorithm.

ISSN No: 2456-2165

Figure 8: Milk Packet Type1

Figure 9: Milk Packet Type 2

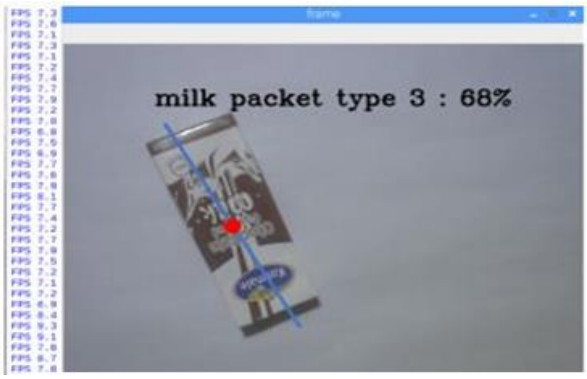

Figure 10: Milk Packet Type 3

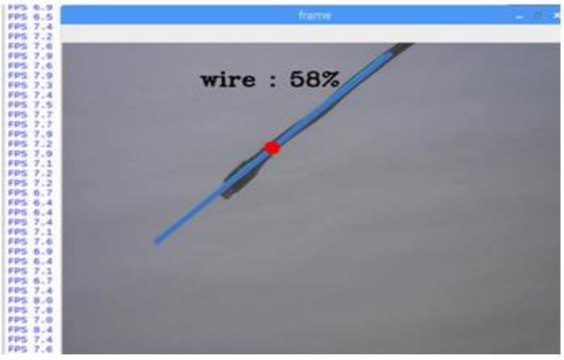

Figure 11: TV Antenna Cable End

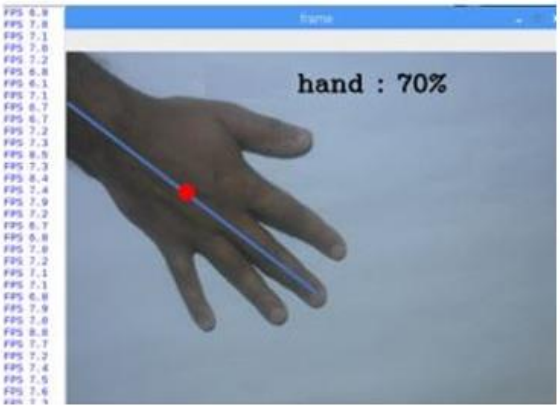

Figure 12: Hand

IV. CONCLUSION

Yolo algorithm is one of the best algorithms for real-time object detection. Yolo-Lite algorithm proposed in [5] has several architectures which can achieve fps rate better than 10 on non-GPU based computers. Detection of objects with 10 fps rate is sufficient for many real-time applications. We studied the achievable speed of Yolo-Lite algorithms on Raspberry PI. As the performance is poor for real-time applications, we presented a new architecture which can achieve a fps rate close to 10. This new architecture is used as a real-time vision system for an industrial robot and shown promising results.

REFERENCES

[1]. J. Schmidhuber, "Deep learning in neural networks: An overview," Neural networks, vol. 61, pp. 85–117, 2015.

[2]. Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

[3]. Redmon, J. and Farhadi, A., 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7263-7271).

[4]. Souza, R., Freitas, M., Jimenez, M., Magalhães, J., Kubrusly, A.C. and Rodriguez, N., 2020. Real-time performance assessment using fast interrupt request on a standard Linux kernel. *Engineering Reports*, *2*(1), p.e12114.

[5]. Pedoeem, J. and Huang, R., 2018. YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers. arXiv preprint arXiv:1811.05588.

[6]. J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," arXiv preprint, 2017.