CPU Based YOLO: A Real Time Object Detection Algorithm

Md. Bahar Ullah
Department of Electrical and Electronic Engineering
University of Chittagong
Chittagong, Bangladesh
mdbahar148@gmail.com

***Abstract*—This paper describes CPU Based YOLO, a real time object detection model to run on Non-GPU computers that may facilitate the users of low configuration computer. There are a lot of well improved algorithms for object detection such as YOLO, Faster R-CNN, Fast R-CNN, R-CNN, Mask R-CNN, R-FCN, SSD, RetinaNet etc. YOLO is a Deep Neural Network algorithm for object detection which is most fast and accurate than most other algorithms. YOLO is designed for GPU based computers which should have above 12GB Graphics Card. In our model, we optimize YOLO with OpenCV such a way that real time object detection can be possible on CPU based Computers. Our model detects object from video in 10.12 – 16.29 FPS and with 80-99% confidence on several Non –GPU computers. CPU Based YOLO achieves 31.05% mAP.**

Keywords—object detection, real time, YOLO, CPU, deep learning

I. INTRODUCTION

Computer vision is associate knowledge base field that has been gaining enormous amounts of traction within the recent years and self-driving cars have taken center stage. Another integral part of computer vision is real time object detection. Real time object detection aids in cause estimation, vehicle detection, traffic control, CCTV monitoring etc. For this reason, day to day progress in technology is necessary so that the ability to give vision in computer is possible. So, low cost technology and learning tools are in requirement [1], [2], [3].

The techniques which are currently used for computer vision are more costly than previous because of developing larger and deeper networks to gain higher accuracy [4], [5], [6]. Techniques that are entirely computer based mostly need vast quantity of GPU power and even then aren't always real time, making them inappropriate for day to day applications.

The methods such as YOLO(You Only Look Once) [7], Faster R-CNN [8], Fast R-CNN [9] etc. have affluently obtained an efficient, fast and accurate model with high mean average precision (mAP); But, their frames per second (FPS) on non-GPU computers drive them unsuitable for real-time application. In this paper, we are proposing a model named "CPU Based YOLO" to run YOLO [7] algorithm on non-GPU computer with OpenCV [10]. Our model is able to execute videos from external source or from webcam with minimum 10.12FPS, 80-99% confidence and with 31.05% mAP that is suitable for real time application within low cost and less effort.

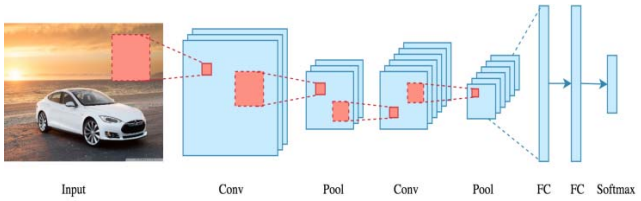


Fig. 1. Convolutional Neural Networks design

II. RELATED WORK

The evaluation in computer vision with Deep Learning has been established and accomplished with time, primarily over one particular well-known algorithm named Convolutional Neural Networks (CNN). Fig. 1 shows the architecture of CNN. It has a convolution layer where a filter is convolved with various parts of the input to generate the output. The use of a convolution layer permits for relative patterns to be drawn from associate input. Moreover, a convolution layer tends to own less weight that require to be learned than a completely connected layer as filters don't need assigned weight from each input to each output [1], [2].

A. YOLO

To generate a single step procedure including object detection and classification You Only Look Once (YOLO) [7] was introduced. YOLO is successful to gain 45 FPS and further a lite version named Tiny-YOLO, achieves around 244 FPS (Tiny-YOLOv2 [11]) on GPU computers.

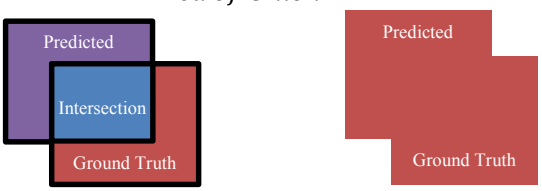
Why YOLO is differ from other existing networks? The answer is: at the same time bounding box predictions and class predictions can be done by YOLO. First, the input image is sliced up into $S \times S$ grids [7]. Second, B bounding boxes are included in each grid cell, each with a score of confidence. The probability of an object exists in each bounding box is known as confidence and is defined as:

$$C = Pr(Object) * IOU_{pred}^{truth} \quad (1)$$

Where, IOU is the abbreviation of intersection over union which describes a measure (a fraction between 0 and 1) of overlap between two bounding box. Intersection is the overlapped area between the predicted bounding box and ground truth (Object), and union is the total area of predicted and ground truth boxes as drawn in Fig. 2. For accurate detection, the IOU should be near to 1, so that the predicted bounding box lies closer to the ground truth. IOU can be expressed as:

978-1-7281-7366-5/20/$31.00 ©2020 IEEE

$$IOU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \quad (2)$$



(a) Area of Overlap (b) Area of Union

Fig. 2. Intersection Over Union

At the same time, while generating bounding boxes, each grid cell predicts C conditional class probability of the object. The class-specific probability for each grid cell is:

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth} \quad (3)$$

YOLO utilizes the equation below for calculating loss function and finally improve confidence [7]:

LossFunction:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$
$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(C_i - \hat{C}_i\right)^2$$
$$+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(C_i - \hat{C}_i\right)^2$$
$$+ \lambda_{noobj} \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2. \quad (4)$$

To legitimate the center and the bounding box of each prediction, the loss function is used. Every input image is divided into an $S \times S$ grid, with B bounding boxes for each grid. The x_i and y_i variables are coordinates of center of each prediction, while w_i and h_i refer to dimensions of bounding box. The λ_{coord} variable is used to increase emphasis on boxes with objects, and λ_{noobj} variable decrease the emphasis on boxes with no objects. C refers to the confidence score, and $p_i(c)$ refers to the prediction of classification. The 1_{ij}^{obj} is 1 if the j^{th} bounding box in the i^{th} cell predicts the object, an else 1_{ij}^{obj} is 0. If the object is in cell i then 1_i^{obj} is 1 and 0 otherwise. The loss function indicates the performance of the model, if loss is low then performance will be high [7]. The accuracy of predictions generated by models in object detection is calculated through the average precision equation defined as [7]:

$$avgPrecision = \sum_{k=1}^{n} P(k)\Delta r(k) \quad (5)$$

$P(k)$ refers to the precision at threshold k and $\Delta r(k)$ refers to the fluctuate in recall [7].

TABLE I. COMPARISON BETWEEN DIFFERENT YOLO VERSIONS

Model	Performance Evaluation				
	Input Image Size	*Train Set*	*Test Set*	*mAP*	*FPS*
YOLOv1	448x448	VOC 2007+2012 [13]	VOC 2007	63.4	45
Fast YOLOv1	448x448	VOC 2007+2012	VOC 2007	52.7	155
YOLOv2	416x416	VOC 2007+2012	VOC 2007	76.8	67
Tiny – YOLOv2 [11]	416x416	VOC 2007+2012	VOC 2007	57.1	207
YOLOv2	608x608	COCO [14]	COCO	48.1	40
YOLOv3 [12]	608x608	COCO	COCO	57.9	20

YOLO is later upgraded with various versions such as YOLOv2 or YOLOv3[12] in order to optimize localization errors and increase mean average precision (mAP). The FPS and mAP of different version of YOLO is shown in Table I.

B. Frameworks

To develop our model we need to install a deep learning framework where we will run the YOLO algorithm. There are few frameworks for running an algorithm which are discussed below:

- TensorFlow: It is a deep learning framework created by Google which can be used for designing, building, and training models. But it needs huge amount of GPU power and is favorable to Linux [15].
- Darknet: It was developed by the developer of YOLO based on Linux. And it runs better on GPU based computers [16].
- Darkflow: It was made by adapting darknet to Tensorflow and works very fast in GPU based computers. It also runs in CPU based computers but installing in windows is terrible and very slow [17].
- Opencv: It was built by Intel and also it has a deep learning framework. It works only in CPU and installation in windows is easy [10].

III. CPU BASED YOLO ARCHITECTURE

Our aim is to build the model (CPU Based YOLO) for CPU Based real time object detection. Developers of YOLO used Darknet [16] framework for running the algorithm. We run YOLOv3 with DarkFlow [17] and OpenCv [10] as only they are favorable for CPU Based YOLO. Our aim was to develop the model in Windows operating system but we found that DarkFlow installation in windows is really a complex task. However, we installed it and run YOLOv3 [12] on it, but the result was terrible. The FPS was too low and the starting time was too much. Then we start to do the task on OpenCv. We input a video though OpenCv and found a good FPS in the output. The architecture is shown in Fig. 3.

A. Setup

We loaded YOLOv3 and Dataset (COCO) [14] through the framework OpenCv. Procedure of loading programme is shown in Fig.4.

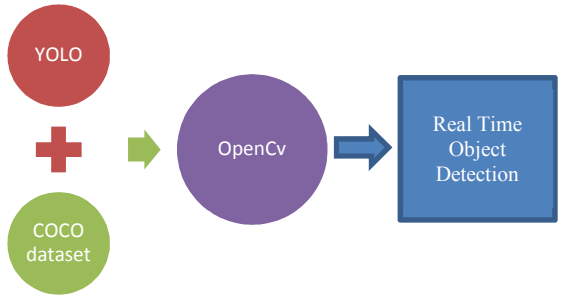


Fig. 3. Architecture of CPU Based YOLO

When we input a video then we found that the system was detecting object from the input but the FPS was too low (around 2FPS) but higher than that of DarkFlow. Then we focused on increasing FPS and began to reduce the Blob size in the input video and observed the detecting performance for testing. The loading programme is shown in Fig. 4.

```
# Load Yolo
net = cv2.dnn.readNet("weights/yolov3.weights", "cfg/yolov3.cfg")
classes = []
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]
layer_names = net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
colors = np.random.uniform(0, 255, size=(len(classes), 3))
```

Fig. 4. Programme of Loading YOLOv3 Using OpenCV

B. Blob Size Reduction

Blob size is a group of connected pixels in the input image or video that assigns several common properties of the object class. We used several blob size for finding perfect FPS in CPU based computers. First, we used blob size 220x220 in the OpenCv [10] function names "cv2.dnn.blobFromImage()", we found 2.40FPS and 99% to 100% confidence in output from webcam video which is above most of the current algorithms. Then we did the same task in several times using the blob size 416x416, 320x320, 128x128, 90x90, 64x64 etc. using several inputs which is shown in Fig. 5. The Computer configuration was AMD Ryzen™ 3 2200G with Radeon™ Vega 8 Graphics 3.50GHz with 8GB RAM, shown in Fig. 6. We found that low blob size increase the FPS but decrease the detection accuracy.

```
# Detecting objects
blob = cv2.dnn.blobFromImage(frame, 0.00392, (90, 90), (0, 0, 0), True, crop=False)
```

Fig. 5. Blob Size Reduction Programme

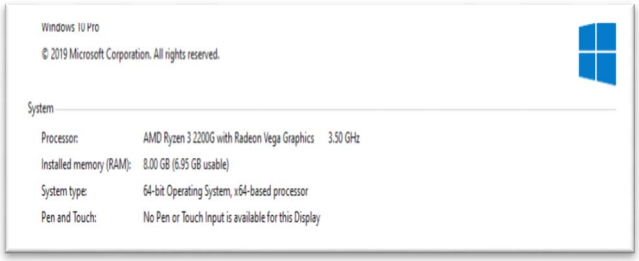
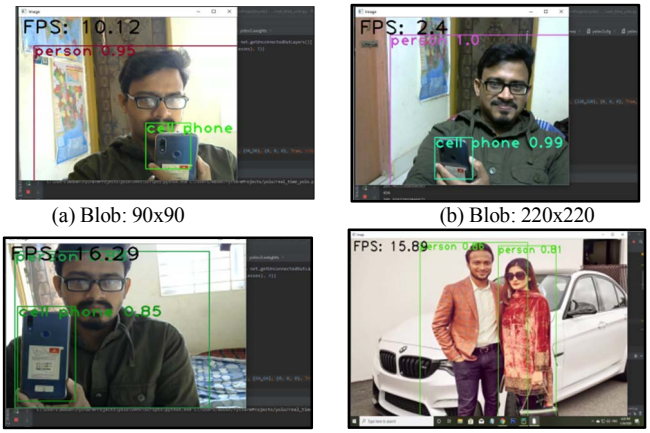


Fig. 6. AMD CPU Used for Experiment -1

But we need an optimum value of blob size for CPU Based YOLO. We found that in 90x90 the detection was good accurate and also the FPS is above 10. In 64x64, the FPS is above 16 but not so accurate for tiny objects like cup, book, cellphone. The result is shown in Fig. 7. We run the same task in many videos and found almost same result. As we used OpenCv, GPU was not used by the framework.



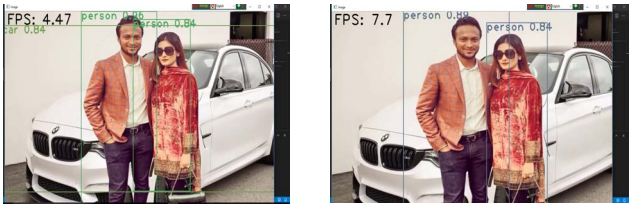
(a) Blob: 90x90 (b) Blob: 220x220

(c) Blob: 64x64(not detecting the car)

Fig. 7. FPS and Confidence in Different Blob Size from Experiment-1

C. Checking in Another Computer

To justify the system performance in another computer we done the task in a very low configuration laptop which is built by Intel® Core™ i3-5010U CPU @ 2.10GHz with 4GB RAM. We found 4.47FPS when blob size was 90x90 and 7.7FPS when blob size was 64x64, shown in Fig. 8. This FPS is also higher than most other real time object detection models.



Blob: 90x90 Blob: 64x64

Fig. 8. FPS and Confidence in Different Blob Size from Experiment-2

D. mAP calculation

We perform mAP calculation in COCO dataset [14] in our computer using OpenCV, JSON [20], Matplotlib and found 31.05% mAP, shown in Fig. 9. The mAP is low due to reducing the blob size of the input video. But it will not be a big fact for low configuration computer users.

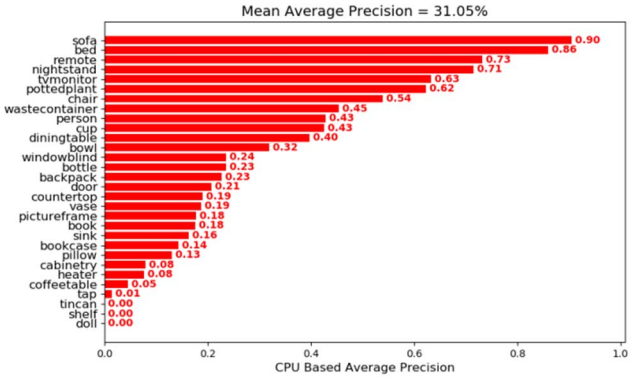


Fig. 9. mAP from Experiment

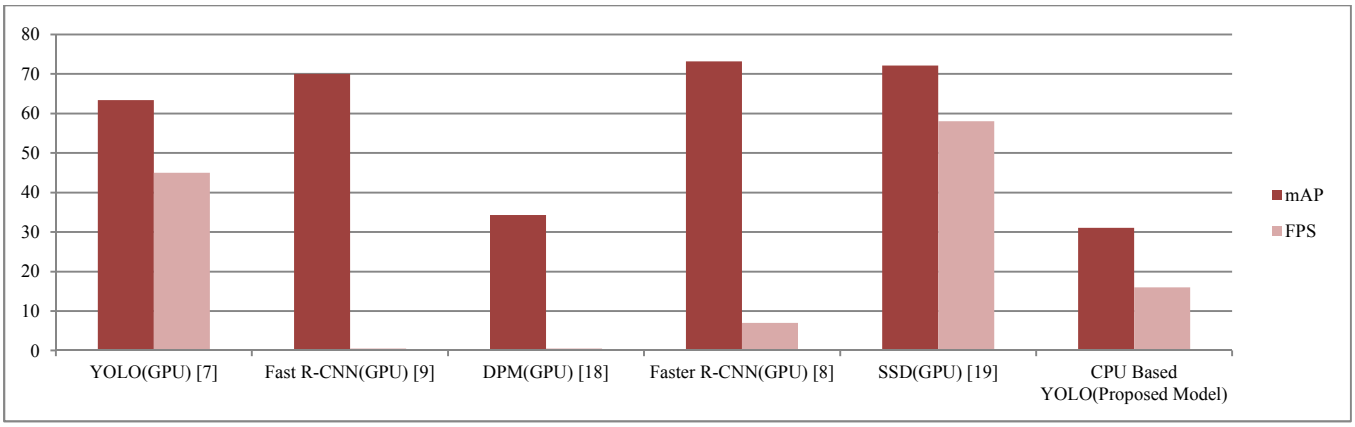


Fig. 10. Comparison with other Networks

IV. Results and Comparison With Other Algorithms

We perform several experiments in several computers for finding an optimum value of blob size so that we can do the task with better accuracy and FPS. We found 31.05% mAP and 16FPS (maximum) which is good enough for low configuration CPU based computers. The results for the experiments are shown in the Table II. From the comparison with other GPU based algorithms, it is clear that our model is eligible for CPU based applications. The comparison is shown in Fig. 10.

TABLE II. Results from experiments on Two CPU

Computer /Laptop	RAM	Dataset	mAP	FPS
AMD Ryzen™ 3 2200G CPU 3.50GHz	8GB	COCO [14]	31.05%	16
Intel® Core™ i3-5010U CPU @ 2.10GHz	4GB	COCO	31.05%	7.7

V. Conclusion and Future Work

CPU Based YOLO gains the purpose of real time object detection on Non - Computers. First, we selected a right framework for doing object detection task in CPU then we utilized YOLO algorithm in a right manner and done the task successful. We can apply this model to normal Desktop or Laptop for executing YOLO. Thus we can do real time object detection for several purposes like video surveillance, traffic monitoring, face tracking etc. In future, we will perform the tasks by training the machine with custom dataset. We will increase the mAP and FPS by optimizing the model.

REFERENCES

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

[2] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in Advances in neural information processing systems, 2013, pp. 2553–2561. 1

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580–587. 1

[4] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2818–2826. 1

[5] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." In AAAI, vol. 4, 2017, p. 12. 1

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778. 1

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788. 1, 2

[8] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," CoRR, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497. 1

[9] R. B. Girshick, "Fast R-CNN," CoRR, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083. 1

[10] G. Bradski, A. Kaehler, 2008, Learning OpenCV – Computer Vision with the OpenCV Library, O'Reilly Media Inc. Sebastopol. 1, 2, 3

[11] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," arXiv preprint, 2017. 2

[12] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," arXiv preprint arXiv:1804.02767, 2018. 2

[13] M. Everingham, L. Van Gool, C. Kl. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. JJCV, 2010. 2

[14] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár. Microsoft COCO: Common Objects in Context. In ECCV. 2014. 2, 3, 4

[15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016, arXiv:1603.04467. 2

[16] J. Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016. [Accessed January 06, 2020]. 2

[17] Darkflow, 2019. https://github.com/thtrieu/darkflow. [Accessed December 25, 2019]. 2

[18] P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. IEEE Transactions on Pattern Analysis and Machine Intelligence, (9):1627–1645, Sept 2010.4

[19] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.- Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In European conference on computer vision, pages 21–37. Springer, 2016. 4

[20] C. Severance, "Discovering JavaScript Object Notation," Computer, vol. 45, pp. 6–8, 2012.4