Multi-Thread Approach to Object Detection Using YOLOv3

Rayan Abri*, Sara Abri*, Anıl Yarıcı*and Salih Çetin*
* Mavinci Informatics Inc., Ankara, Turkey
Email:rayan.abri@mavinci.com.tr
sara.abri@mavinci.com.tr
anil.yarici@mavinci.com.tr
salih.cetin@mavinci.com.tr

***Abstract*—Recently, In the field of the object detection process, Convolutional Neural Network (CNN) methods have been applied as more efficient solutions. In the CNN methods, You Look Only Once, version 3 (YOLOv3) more powerful than classic CNN methods such as Fast-RCNN and Faster-RCNN in terms of accuracy and speed. Even though YOLOv3 can obtain more accuracy and speed compared to other CNN approaches, it needs to be used in a system with a powerful single Graphics Processing Unit (GPU). The YOLOv3 still requires the heavy computational cost to maintain good detection performance; it brings high computation overhead. It is needed to design a more stable algorithm to increase frames per second (fps) in the object detection process. The main contribution is proposing a Multi-thread approach that uses YOLOv3 to perform real-time object detection in the large scale of video streams. In this paper, we design a Multi-thread approach that uses YOLOv3 to perform real-time object detection with decreasing detection time per frame. In the proposed multi-thread approach, we aim to decrease the idle times between CPU and GPU. The proposed approach is evaluated using a public dataset and the result shows improvements in performance compared to the YOLOv3. The multi-thread model is evaluated on two different servers with two GPU cards and the results yield improvement by an average 26% in fps.**

***Contribution*—The main contribution is a Multi-thread approach that uses YOLOv3 to perform real-time object detection in the large scale of video streams.**

***Index Terms*—convolutional neural network, fast-RCNN, faster-RCNN, YOLOv3, multi-thread**

I. Introduction

Recent research in the field of object detection based on convolutional neural network (CNN)-methods such as Region-CNN (R-CNN) [1], Faster R-CNN [3], YOLO [2], YOLOv2 [4] and YOLOv3 [5] have been shown more efficient than other detection algorithms which focus on making traditional detection [8], [11], [12]. Generally, in object detection methods, the steps of extraction feature of images [7], [9], classification [13]–[15] and localization [6], [10] are used.

YOLO was proposed by Joseph Redmon [2] and passes the n by n image only once in a convolutional neural network (CNN), which makes it quite fast in real-time. In the proposed YOLO approach, the object detection process is formulated by computing bounding box coordinates and class probabilities at the same time. In this way, the problems related to computational complexity in RCNN are overcoming. Although YOLO was demonstrated to provide significant speed advantages over R-CNN, it was also shown that the localization error of YOLO is higher than more recent RCNN such as Faster RCNN.

Region proposal network (RPN) in Faster R-CNN predicts offsets and confidences instead of predicting bounding box coordinates directly. Each anchor is associated with 4 coordinates for the box and 2 score values which estimate the probability of object and not object of the proposed box. In [2] proposed an improved YOLO method (named YOLOv2) where anchor boxes are used to predict bounding boxes.

Furthermore, compared to YOLO network architecture, YOLOv2 does not have fully-connected layers. YOLOv2 make a faster network by utilized a new CNN network architecture based on Darknet-19. The experimental results for this approach demonstrated that YOLOv2 can perform object detection at 67 FPS on a Nvidia Titan-X GPU while achieving detection performance. YOLOv2 [4] overcomes the high localization error and low recall, as compared to region-based techniques. Although YOLOv2 can achieve real-time performance on a powerful GPU, it remains very challenging for leveraging this approach for real-time object detection in the video on embedded computing devices with limited computational power and limited memory.

In 2018, YOLOv3 [5] is released and is characterized by higher accuracy and replaces softmax function with logistic regression and threshold. The YOLOv3 uses YOLOv2 as a base structure along with 53 convolutional layers. YOLOv3 is more powerful and faster than YOLOv2 because of using GPU as more efficient. YOLOv3 can obtain more accuracy and speed compared to other approaches but it is needed to use a powerful Graphics Processing Unit (GPU) to execute.

In different real-world applications such as concurrent real-time inference on a GPU server in a commercial system, the available computing GPU resources are limited memory. YOLOv3 has low performance in terms of object detection time per frame in normal GPU. To solve the problem, in this paper a model is proposed which uses YOLOv3 for concurrent real-time objection detection on a GPU server using multi-thread architecture. Our goal is to optimize the CPU and GPU usage in YOLOv3 and propose a multi-thread model for

978-1-7281-9331-1/20/31.00 ©2020$IEEE$

concurrent real-time object detection in multiple video stream scenarios.

Our purpose is to provide an optimized architecture that has significantly increase the fps. Therefore, this architecture makes possible use of multi-thread YOLOv3 on a GPU server with high speed in detection performance. We have implemented the proposed model on the Nvidia Quadro p5000 and Nvidia GTX 1070ti by the CUDA architecture. The result shows improvements in performance by an average of 13% in fps compared to the YOLOv3. In Section II, related works are discussed. The methodology is presented consists of improvement in Section III. In Section IV, the experimental result is presented. The conclusion is drawn in Section V.

II. Related Work

There are some important improvements for applications such as image classification, object detection [31], face detection [24], segmentation [32] and object tracking [19], [33] using convolutional neural networks. The traditional method for object detection focused on SIFT such as the Fast Point Feature Histograms (FPFH) [20] and Normal Aligned Radial Features (NARF) [21] that are used in 3D image registration. Classification methods are also used for object detection include nearest-neighbor methods [22] and support vector machines [23].

In the convolutional neural networks (CNN) methods [30], there are improvements for object detection such as RCNN, Fast-RCNN, and Faster-RCNN [3]. More recent methods using CNN such as YOLO [2], YOLOv2 [4], YOLOv3 [5] and Mvc-YOLO [25], [36]. Object detection approaches have been depicted more improvement in comparison with traditional computer vision methods such as methods based on SIFT [16]. Using deep learning-based strategies [34], they provide significant speed advantages over R-CNN (for example 45 frames per second in YOLO on an Nvidia Titan-X GPU). Although convolutional neural networks require training process they can be applied for general challenges. They also can be used along with fewer hardware resources.

There is a gap between software and hardware implementations [35] due to high power consumption. In fact, there is a need to implement hardware along with an efficient neural network design in order to exploit CNNs for low-power. There are some researches on real-time processing using multi-core architectures and graphics processing units (GPUs). The methods such as Gaussians mixture model (GMM) for background modeling are used based on such architectures in reference [17]. In GPU architecture, NVIDIA has provided GeForce, Quadro and Tesla/Fermi series with a different range of performance. In [26] a hardware architecture for real-time object detection is proposed that using depth and edge information. In [27] an analytical framework (OPTiC) is proposed for partitioning optimal CPU-GPU co-execution on systems. In [28] a GPU-based floating real-time object detection system is proposed. Based on the results in this paper, using GPU compared to CPU can increase speed and improve performance. Using Convolutional Neural Networks, in other applications like IoT and mobile edge computing is discussed in [29]. In the paper real-time multiple object tracking is implemented on an NVIDIA.

III. Methodology

The proposed methodology is divided into two improvement phases. In the first phase, it is aimed to optimize the architecture of the YOLOv3 with eliminating the unnecessary computations and conversions and in the second phase, it is aimed to propose the optimized multi-thread model to increase performance in terms of fps for handling multiple video streams.

In the first improvement, it is planned to explain the main architecture of the object detection process. The main architecture of the YOLOv3 is depicted in Figure 1. This architecture including the object detection process using YOLO on the Darknet framework. As shown in the figure, there are computing and converting processes. All of the computing processes are done on the GPU side and converting processes are conducted on the CPU side. The process is started with converting RGB image as a 3D matrix to the YOLO image using a function. The YOLO image is a 2D matrix contains RGB values for each pixel. The main function of object detection is done using YOLO on a converted YOLO image. The output of the detection is a vector of coordinates for each detected object.

As previously mentioned, YOLOv3 uses a convolutional neural network to detect objects. Since in this paper, our goal is to discuss the architecture of the object detection process, so we avoid the explanation of the YOLOv3 method in detail. Finally, the CPU converts the YOLO image to RGB image format. Although the YOLOv3 can obtain more accuracy and speed compared to other approaches but YOLOv3 needs to be used in a system with a powerful single Graphics Processing Unit (GPU).

In addition to there is unnecessary computations and conversions in the object detection process in Darknet framework. This paper provides an improved model to optimize the process of the detection using YOLO in Darknet framework as shown in Figure 2.

The presented model changes detected coordinates using YOLO detection algorithm and eliminate the unnecessary conversion at the last step of the Darknet framework. For each detected object there is coordinate point which describes the center of each object and two scales as the percentage of the specified object's length and width. In this step we intervene the normal process of YOLO and calculate new coordinates instead of drawing detected regions by using YOLO provided coordinates. For calculating new coordinates, we obtain the left corner, length and width of the detected region using the information of provided coordinates by YOLO. Our purpose is to draw regions obtained from detected objects on the RGB image directly.

In fact, YOLO make additional computations with drawing detected regions on the YOLO image and then unnecessary conversions YOLO image to RGB image. Therefore, for

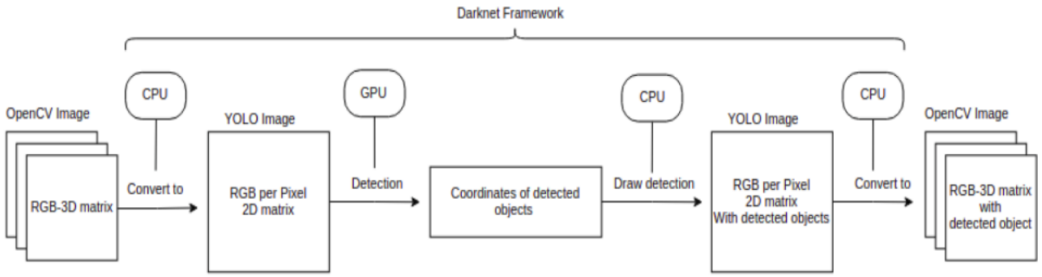

Fig. 1. The architecture of the object detection process in YOLOv3.

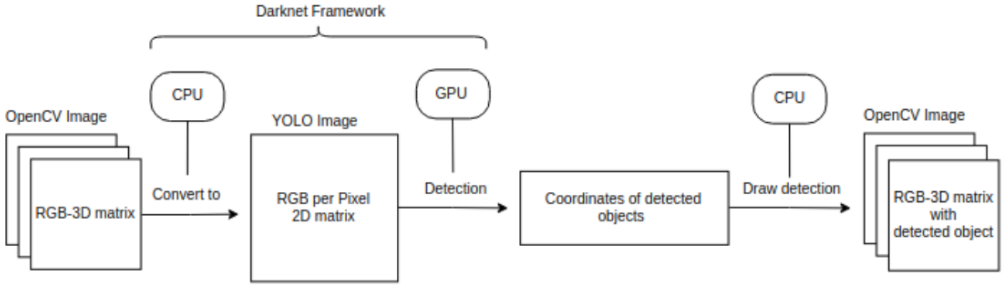

Fig. 2. The architecture of the Presented model.

achievement the more efficiency we prevent drawing regions by YOLO on YOLO image and re-draw regions using new estimated coordinates directly on the OpenCV RGB image. Finally, we consider new coordinates to draw regions on RGB image directly using estimated coordinates. These extra calculations and conversions may be considered little but we obtain improvements in term of CPU usage by eliminating them.

In the second improvement, this paper provides a multi-thread approach with decrease CPU and GPU idle times to improve performance in terms of fps. In original YOLOv3, frames are converted from RGB Image to Yolo Image in the CPU side and the detection process is performed on the GPU side as directly without the use of multi-thread methods. There are time intervals that system resources are idle. Therefore we aim to design an optimal multi-thread model to reduce the idle times or gaps of time slots in the CPU and GPU side individually.

In Figures 3 and 4, the multi-thread model is shown using pseudo-code on the CPU and GPU side processes. Proposed Multi-thread model starts with buffering step, The threading process in the Multi-Thread model starts by buffering the video frames and then sends them to the CPU and GPU separately. In the buffering process, the video frames are placed in a queue to order and preparation for sending. It is explained pesudo code of multi-thread approach in two algorithms. In the first algorithm, frames and next_frame keep buffered frames. The reason existence of next_frame is the buffering system must be nonstop. The second one is the CPU side processes (ParallelConvert function). The video frames are sent using a ParallelConvert function as defined in Figure 3. This function performs the conversion process from RBG Image to Yolo Images. It uses different cores of CPU to implement threading work. The third one is GPU side processes (Detect function). The main work of detection is done using a Detect function that performs object detection on the GPU side as a multi-thread.

IV. Evaluation Methodology and Experiments

In this section, we evaluate the methodology and experiments in two sub-sections. The first subsection discusses the experiment environment. The second subsection presents experiments to evaluate the proposed model and approach in Section III by describing the architecture structure.

A. Experiment Environment

In this section, we evaluate the performance of the proposed model and architecture in term of detection time or fps. According to the results obtained from experiments, the accuracy of the proposed method in this paper is the same as the YOLOv3 accuracy. We tested the different streams of the video with different resolutions, and the results showed that there was no change in accuracy. We prepare an experiment to compare the efficiency between the YOLOv3 and the Multi-thread approach. Our evaluation is on a dataset of 2D MOT 2015 containing videos with fps 30 and different qualities resolution (1280×720, 1920×1080 and 3840×2160 pixels). This benchmark contains video sequences in different environments. The properties of video analysis servers used as a test server are presented in Table I.

TABLE I
THE PROPERTIES OF VIDEO ANALYSIS SERVERS.

Parameters	Video Analysis Server 1	Video Analysis Server 2
CPU	Intel core i9-7940X 14 core/ 28 thread	Intel core i9-7940X 14 core/ 28 thread
GPU	Nvidia Quadro p5000 16GB	Nvidia GTX1070ti 11 GB
RAM	32 GB DDR4	32 GB DDR4
Disk	256 NVM Express SSD	256 NVM Express SSD
Environment	Ubuntu 18.04 LTS	Ubuntu 18.04 LTS

```
Algorithm 1: Multi-Thread Model
  StartProccess(video_source)
    frames = Buffer(video_source)
    next_frames = Buffer(video_source)
    thread = BeginThread(Detect, frames)
  while !at_the_end_of_video do
      detections = EndThread(thread)
      thread = BeginThread(Detect, next_frames)
      posterior_frames = [frames.size]
      while frames.size do
          show(frames[i], detections[i])
          posterior_frames[i] = Buffer(video_source)
      end
      frames = next_frames
      next_frames = posterior_frames
  end
    detections = EndThread(thread)
```

Fig. 3. Psudo code of Multi-thread model.

```
Algorithm 2: CPU and GPU Processes
  Detect(frames)
      yolo_images = ParallelConvert(frames)
      detector = ChooseDetectorIsIdle()
      rects = GetObjectBoxes(detector, yolo_images)
      return rects
  ParallelConvert(frames)
    threads = []
  while frames.size do
      show(frames[i], detections[i])
      thread = []() →
              return ConvertQImagesToYoloImages(frames[i])
      threads.Append(thread)
  end
  WaitForAllThreads(threads)
  return ArrayReturnValues(threads)
```

Fig. 4. Psudo code of CPU and GPU processes.

B. Experimental Evaluation of the Model

As discussed before, to evaluate the proposed multi-thread model, we make an experimental comparison between the Original YOLOv3 and the Multi-Thread YOLOv3 model. We run 30 video streams in each video resolution on both of the models and the detection time measurement based on fps with the different resolution are compared in the original YOLOv3 and the proposed Multi-Thread approach. The Gaussian distributions of YOLOv3 and the proposed multi-thread model in 720p resolution are shown in Figure 5. To find out which method is more stable in terms of detection time (1/f ps), we need to compare two histograms.

As shown in Figure 5, the histogram of the Multi-Thread YOLOv3 model has a smaller variance than the original YOLOv3. It indicates that the Multi-Thread YOLOv3 model has more stable fps or detection time compared to the original YOLOv3.

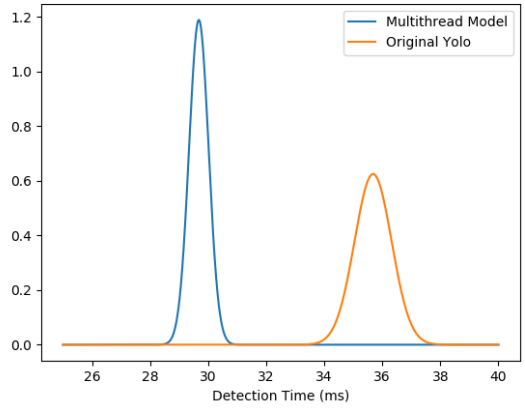

Fig. 5. Histogram Comparison of the Original YOLOv3 and the Multi-Thread YOLOv3 model.

The results are presented in Figure 6 in terms of fps. Based on the results, it can be seen that the video frames per second of the original YOLOv3 and the Multi-Thread model has decreased with increasing the video qualities and the Server 1 is faster than the Server 2 because of CUDA core size. The CUDA core size in the Server 1 is 2560, it is while, the Server 2 has 1920 CUDA core size. The result shows that the detection algorithm with the Multi-Thread model is approximately 34% faster than the original Yolov3 in all tested video resolutions of 480p, 720p, and 1080p.

V. CONCLUSION

In this paper, we present a multi-thread model to increase the performance of the YOLOv3. The YOLOv3 is one of the most popular object detectors in practical applications as the detection accuracy and speed are well balanced. Despite that, YOLOv3 still requires heavy computational cost and it brings

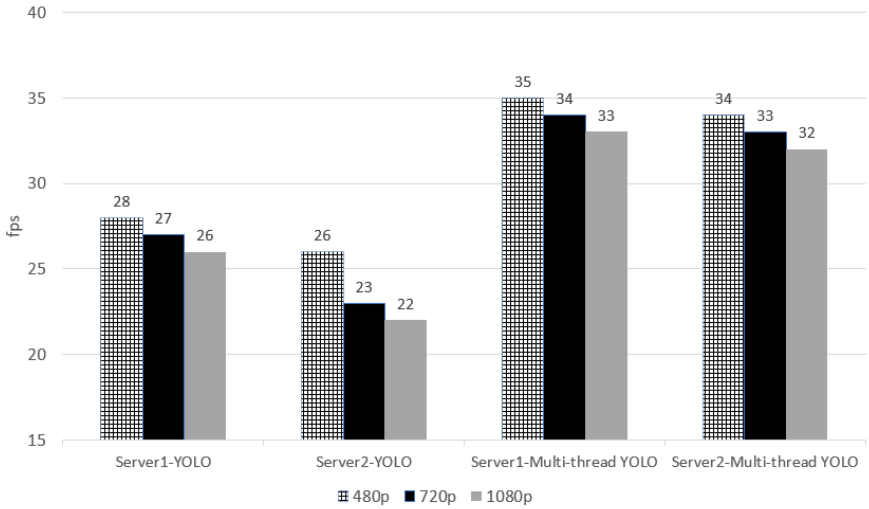

Fig. 6. Performance comparison between the YOLOv3 and the Multi-Thread model.

high computation overhead. This paper proposes a stable and fast multi-thread model to increase performance in terms of fps. Experimental results show that the proposed approach can increase performance by an average of 34% in fps compared to the original YOLOv3.

ACKNOWLEDGMENT

This research is supported by Mavinci Informatics Inc. in Turkey. Mavinci is an R&D company working especially in information and communication technologies, security and defense areas with the capability of software development, artificial intelligence, and machine learning.

The operational areas are; Intelligent Video Analytics based on Deep Learning, Nuclear Safety Research and Analysis, Disaster and Emergency Management, Decision Support Systems, Command and Control Systems, Chemical Biological Radiological and Nuclear Security Solutions, Image Processing and Project Management.

REFERENCES

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," CVPR'14 Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587, 2014.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol.1, pp. 779–788, 2016.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R- CNN: Towards real-time object detection with region proposal networks," Advances in neural information processing systems (NIPS), vol. 39, pp. 1137–1149, 2015.

[4] J. Redmon, and A. Farhadi, "YOLO9000: better, faster, stronger," Computer Vision and Pattern Recognition (CVPR), vol. 1, no. y, pp. 6517–6525, 2017.

[5] J. Redmon, and A. Farhadi, "YOLOv3: An Incremental Improvement," Technical report, 2018.

[6] M. B. Blaschko, and C. H. Lampert, "learning to localize objects with structured output regression," In Computer Vision ECCV, pp. 2–15, 2008.

[7] N. Dalal, and B. Triggs, "Histograms of oriented gradients for human detection," In Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society Conference on, vol. 1, pp. 886–893. IEEE, 2005.

[8] T. Dean, M. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik, "Fast, accurate detection of 100,000 object classes on a single machine," In Computer Vision and Pattern Recognition (CVPR), IEEE Conference on, pp 1814–1821, 2013.

[9] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng et al, "Decaf: A deep convolutional activation feature for generic visual recognition," vol. 32, pp. 647–655, 2013.

[10] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," In International Conference on Learning Representations (ICLR2014), CBLS, abs/1312.6229, 2013.

[11] M. A. Sadeghi, and D. Forsyth,"30hz object detection with dpm v5," In Computer Vision ECCV 2014, pp. 65–79, Springer, 2014.

[12] J. Yan, Z. Lei, L. Wen, and S. Z. Li, "The fastest deformable part model for object detection," In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pp. 2497–2504, IEEE, 2014.

[13] B. Benjdira, T. Khursheed, A. Koubaa, A. Ammar, K. Onuni, "Car Detection using Unmanned Aerial Vehicles: Comparison between Faster R-CNN and YOLOv3," Proceedings of the 1st International Conference on Unmanned Vehicle Systems (UVS), vol. 1, pp. 5386–9368, 2019.

[14] 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, vol. 32, pp. 1627–1645, 2010.

[15] P. Viola, and M. Jones, "Robust real-time object detection," International Journal of Computer Vision, vol. 4 pp. 34–47, 2001.

[16] D. Lowe, "Object recognition from local scale-invariant features," The Proceedings of the Seventh IEEE International Conference on Computer Vision, vol. 2, pp. 1150–1157, 1999.

[17] P. Kumar, A. Singhal, S. Mehta, and A. Mittal, "Real-time moving object detection algorithm on high-resolution videos using GPUs," Journal of Real-Time Image Processing, vol. 11, no.1, pp. 93–109, 2016.

[18] R.T. Collins, A. J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin et al, "VSAM: a system for video surveillance and monitoring," Technical Report CMU-RI-TR- pp. 00-12, Carnegie Mellon University, Pittsburgh, PA 2000.

[19] S. Veeraraghavan, and A. Chellappa, "Object detection, tracking and recognition for multiple smart cameras." Proc. IEEE, vol. 96, no. 10, pp. 1606–1624, 2008.

[20] R. Usu, R. B. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'09), IEEE Press, pp. 1848–1853, 2009.

[21] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, "NARF: 3D range image features for object recognition," In Proceedings of the Workshop on Defining and Solving Realistic Perception Problems in Personal

Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 44, 2010.

[22] J. Liebelt, C. Schmid, and K. Schertler, "Viewpoint- independent object class detection using 3D feature maps," In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08), pp.1–8, 2008.

[23] M. R. Lata, and M. Y. Alvino, "FPGA implementation of support vector machines for 3D object identification," In Proceedings of the 19th International Conference on Artificial Neural Networks: Part I (ICANN'09), Lecture Notes in Computer Science, vol. 5768, Springer-Verlag, pp. 467–474. 2009.

[24] D. Han, J. Choi, J. Cho, and D. Kwak, "Design and VLSI implementation of high-performance face detection engine for mobile applications," In Proceedings of the IEEE International Conference on Consumer Electronics, pp. 705–706, 2011.

[25] S. Abri, R. Abri, A. Yarıcı, S. Çetin, "Multi-Thread Frame Tiling Model in Concurrent Real-Time Object Detection for Resources Optimization in YOLOv3," Proceedings of the 2020 6th International Conference on Computer and Technology Applications (ICCTA '20), pp. 69-73, 2020.

[26] K. Christos, T. Christos, and T. Theocharis, "A Hardware Architecture for Real-Time Object Detection Using Depth and Edge Information," ACM Transactions on Embedded Computing systems (TECS), vol. 13, no. 3, pp. 1–19, 2013.

[27] S. Wang, G. Ananthanarayanan, and T. Mitra, "OPTiC: Optimizing Collaborative CPU-GPU Computing on Mobile Devices with Thermal Constraints," IEEE transactions on computer-aided design of integrated circuits and systems, vol.38 , no. 3 , pp. 393–406, 2018.

[28] Y. Jie, and M. Jian-min, "GPU Based Real-time Floating Object Detection System," 2nd International Conference on Electronics, Network and Computer Engineering 2016.

[29] B. Blanco-Filgueira, D. Garc ia-Lesta, M. Fernandez- Sanjurjo, and P. Lopez, "Deep Learning-Based Multiple Object Visual Tracking on Embedded System for IoT and Mobile Edge Computing Applications," ICDSC '18 Proceedings of the 12th International Conference on Distributed Smart Cameras, No. 22, 2018.

[30] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient- based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, 2016.

[32] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 4, pp. 640–651, 2017.

[33] E. Gundogdu, and A. A. Alatan, "Good features to correlate for visual tracking," IEEE Transactions on Image Processing, vol. 27, no. 5, pp. 2526–2540, 2018.

[34] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proccedings of the IEEE, vol. 105, no. 12, pp. 2295–2329, 2017.

[35] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," IEEE Micro, vol. 31, no. 5, pp. 717, 2011.

[36] S. Karimi Mansoub, R. Abri, A.H. Yarıcı, "Concurrent Real-Time Object Detection on Multiple Live Streams Using Optimization CPU and GPU Resources in YOLOv3," SIGNAL 2019 : The Fourth International Conference on Advances in Signal, Image and Video Processing, pp. 23–28, 2019.