Concurrent Real-Time Object Detection on Multiple Live Streams Using Optimization CPU and GPU Resources in YOLOv3

Samira Karimi Mansoub
Mavinci Informatics Inc.
Ankara, Turkey
e-mail: samira.karimi@mavinci.com.tr

Rahem Abri
Mavinci Informatics Inc.
Ankara, Turkey
e-mail: rahem.abri@mavinci.com.tr

Anıl Hakan Yarıcı
Mavinci Informatics Inc.
Ankara, Turkey
e-mail: anil.yarici@mavinci.com.tr

***Abstract*— Recently, You Look Only Once, version 3 (YOLOv3) approach has been presented as a more efficient solution in the process of object detection. Despite the fact that YOLOv3 can obtain faster and more accurate results than other approaches, it needs to be used in a system with a single powerful Graphics Processing Unit (GPU). However, sometimes, there is a need to process multiple real-time object detection algorithms concurrently on a single GPU, where each object detection algorithm receives a live stream from a camera. It is challenging to have concurrent object detection from live streams on a single GPU. In this paper, we propose a two-step solution to this problem. In the first step, our goal is to provide a model to optimize memory usage and, in the second step, we propose a multi-thread approach that uses YOLOv3 to perform real-time object detection on multiple, concurrent, live streams on a single GPU. In this approach, GPU resources are optimally used. The proposed approach is evaluated on a public dataset and the result shows improvements in performance by an average of 12% in Central Processing Unit (CPU) usage and 13% in frames per second (FPS) compared to the YOLOv3.**

Keywords-Real-Time Object Detection; YOLO; Multi-Thread Approaches.

I. INTRODUCTION

Recent researches 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 shown improvements when compared with other detection methods which focus on traditional detection [8][11]-[13]. Generally in object detection methods, the following steps are performed: 1) feature extraction of images [7][9], 2) classification [1][14][15] and 3) localization [6][10]. Recent approaches, such as YOLOv3, detect objects using the conventional system with a single GPU. YOLOv3 uses YOLOv2 as a base structure along with 53 convolutional layers. YOLOv3 is more powerful and faster than YOLOv2 because it uses the GPU in a more efficient way.

Although the YOLOv3 is more accurate than other approaches, it needs to be executed in a system with a single powerful GPU. In addition, in some applications, it is desired to use CNN's potential for concurrent, real-time object detection. Hence, an efficient hardware implementation along with an efficient network design are required. It is challenging to leverage this approach for concurrent, real-time object detection. For example, in some real-world applications, such as concurrent real-time inference on a GPU server in a commercial system, the available computing GPU resources are limited in terms of memory. In this case, each object detection approach receives a live stream from a camera and all processing is performed on a GPU. The main problem is system resources such as memory, CPU, and GPU when using them concurrently for real-time detection scenarios.

Alternative methods suggest using multiple GPUs in parallel. To solve this problem, in this paper, a model (network architecture design) is proposed, which uses YOLOv3 for concurrent real-time objection detection on a single GPU server using a multi-thread architecture. Our purpose is to provide an optimized architecture that significantly decreases memory usage while at the same time increasing the number of object detection outputs. Therefore, this architecture makes possible use of multiple YOLOv3s on a single GPU server with low memory usage and high speed in detection performance. We have implemented the proposed model on the Nvidia Quadro p5000 with Linux customized by Nvidia by the Compute Unified Device Architecture (CUDA) architecture. The result shows improvements in performance by an average of 12% in CPU usage and 13% in FPS compared to the YOLOv3.

The paper is organized as follows. In Section 2, related works are discussed. Problem description and methodology are presented in detail in Section 3. In Section 4, the experimental result is presented. The conclusion is drawn in Section 5.

II. RELATED WORK

Convolutional neural networks offer significant improvements for applications such as image classification, object detection [30], face detection [24], segmentation [31] and object tracking [32][19]. Traditional methods for object detection have focused on Scale-Invariant Feature Transform (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 and include nearest-neighbor methods [22] and support vector machines [23].

More recent methods using convolutional neural networks [29] such as YOLO [2], YOLOv2 [4] and YOLOv3 [5] object detection approaches have shown more improvement in comparison with traditional computer vision methods such as methods based on SIFT [16]. Using deep learning-based strategies [33], these methods 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 a training process, they can be applied for general challenges. They can also be used along with fewer hardware resources.

Despite an increase in speed, there is a gap between software and hardware implementations [34] 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 CNN's for low-power. Some research exists on real-time processing using multi-core architectures and GPUs. Methods such as the Gaussians Mixture Model (GMM) for background modeling are used in [17] based on such architectures. In the GPU architecture, Nvidia has provided GeForce, Quadro and Tesla/Fermi series with different performance ranges. In [25], a hardware architecture for real-time object detection using depth and edge information is proposed. In [26], an analytical framework (OPTiC) is proposed for partitioning optimal CPU-GPU co-execution on systems. In [27], a GPU-based floating real-time object detection system is proposed. Based on the results in this paper, using GPU instead of CPU can increase speed and improve performance. Using convolutional neural networks in other applications like Internet of Things (IoT) and mobile edge computing is discussed in [28]. In this paper, real-time multiple object tracking is implemented on an Nvidia GPU architecture.

There has been a very limited number of works on concurrent real-time object detection on multiple live streams. In this paper, we try to provide an optimized architecture to concurrently detect objects on multiple cameras. In this architecture, YOLOv3 is used as a backbone approach to detect objects. We conducted experiments to evaluate the approach. In the next section, we describe the methodology.

III. Problem Description and Methodology

In this section, we discuss two improvements in the performance of the YOLO in regards to the process of object detection. In the first subsection, we present the new architecture model which changes the structure of YOLO in the object detection process to reduce the number of unnecessary computations and conversions in the object detection process. In the second subsection, we provide a multi-thread approach for performing real-time object detection on multiple live streams concurrently. Using these two improvements, we achieve increased efficiency in the object detection process using YOLOv3.

A. The Architecture of the Model

In this subsection, we plan to explain the network architecture of the object detection process. Before we start to discuss the model, we need to describe the architecture of the object detection process in YOLOv3. Then, we discuss the problems of the architecture. Finally, we provide a model as a solution to overcome these problems.

The main architecture of the YOLOv3 is depicted in Figure 1. This architecture includes an object detection process using YOLO on the darknet framework. As shown in Figure 1, there are computing and converting processes. All of the computing processes are done on the GPU side and the converting processes are conducted on the CPU side. The process is started with converting RGB (red, green, and blue) image as a three dimensions matrix to the YOLO image using a function. The YOLO image is a two dimensions matrix that contains RGB values for each pixel. The main function of the 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 conventional neural network to detect objects. Since in this paper our goal is to discuss the architecture of the object detection process, 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, it needs to be used in a system with a powerful single GPU. Also, there are unnecessary computations and conversions in the object detection process in the Darknet framework. Here, we provide an improved model to optimize the process of the detection using the YOLO in the Darknet framework. We call the improved model MvcYOLO. The MvcYOLO model is presented in Figure 3. In this model, we change the 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 a coordinate point which describes the center of each object and two scales as the percentage of the specified object's length and width. These coordinates are produced with YOLO. In this step, we intervene in the normal process of YOLO and calculate new coordinates instead of drawing detected regions by using YOLO provided coordinates. For this propose, we calculate the left corner, length, and width of the detected region using the information of provided coordinates by YOLO. As mentioned before, in YOLO detection, we have the points that belong to the center of each detected object and two percentage of length and width in every detected object. In this process, we convert YOLO detection coordinates to our desired coordinates. Our purpose is to draw regions obtained from detected objects on the RGB image directly. In fact, YOLO makes the additional computations with drawing detected regions on the YOLO image and then unnecessary converts the YOLO image to RGB image. So, to achieve more efficiency, we prevent drawing regions by YOLO on YOLO image and re-draw regions using the new estimated coordinates directly

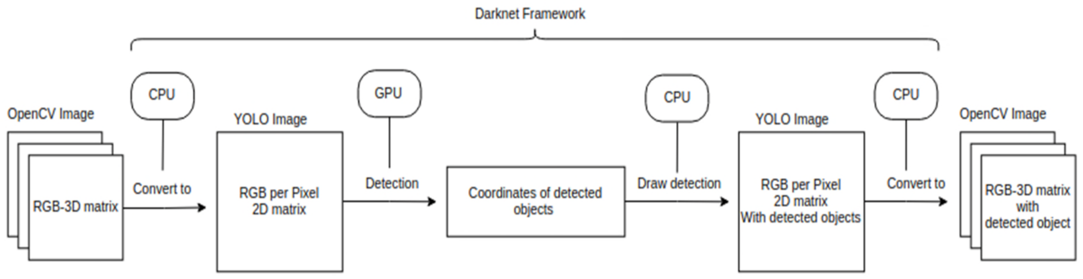

Figure 1. The architecture of the object detection process in YOLOv3.

on the OpenCV RGB image. Finally, we consider new coordinates to draw regions on RGB image directly using the estimated coordinates. These extra calculations and conversions may be considered little, but we obtain improvements in terms of CPU usage by eliminating them. The results are discussed in the next section.

B. Multi-thread Approach to Handle Concurrent Surveillance Videos

The second part of our improvements is focused on the GPU usage per YOLO object. Each YOLOv3 object in the Darknet framework uses 1.70 GB GPU memory. This means that we can handle a surveillance video with 1.70 GB GPU memory. Because the GPU memory is limited in the GPU cards, it is a costly process to cover a large scale of surveillance cameras. This paper provides a multi-thread approach to handle concurrent surveillance videos with each YOLOv3 object. As mentioned before, if each surveillance camera works with a YOLOv3 object, 1.70 GB GPU of memory is needed to be assigned to it. As a result, it requires a lot of GPU memory for performing real-time object detection on multiple live streams concurrently.

To solve this problem, in this paper we present a multi-thread approach as a solution to handle concurrent surveillance videos. In this approach, a pre-processing and a post-processing step are added to the YOLO object. In the pre-processing step, a multiplexer for fragmentation of live streams is applied. In the post-processing also, a de-multiplexer is used to re-fragmentation and assembling the frames of each live stream. An overview of the approach is depicted in Figure 2.

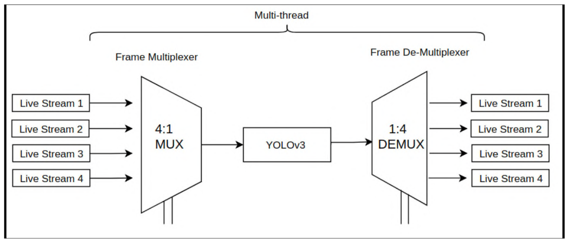

Figure 2. An overview of the multi-thread approach.

As shown in Figure 2, the input and output of YOLO are defined as Y_{In} and Y_{out}. There are a multiplexer and a de-multiplexer at the beginning and at the end of the process. In the multiplexer, in each live stream, a frame of every four frames is selected. Thus, there are intervals of four frames in each live stream. In the next step, the frames are passed to detect objects using YOLOv3. In this way, frames of four live streams are fragmented and are passed separately to the YOLOv3 algorithm to detect the objects. The output of the YOLOv3 algorithm is a video that contains every four video frames. This fragmentation process works as a multi-thread approach. To investigate the efficiency, we evaluated the approach using experiments and the results are presented in the next section.

IV. EVALUATION METHODOLOGY AND EXPERIMENTS

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

A. Experiment Environment

In this section, we evaluate the performance of the proposed model and architecture in the previous section. First, we prepare an experiment to compare the efficiency between the YOLOv3 and MvcYOLO. Our evaluation is on a dataset of 2D MOT 2015 containing 30 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 a video analysis server used as a test server are presented in Table I.

TABLE I. THE PROPERTIES OF VIDEO ANALYSIS SERVER.

CPU	Intel core i9-7940X 14 core/ 28 thread
GPU	Nvidia Quadro p5000 16GB
RAM	32 GB DDR4
Disk	256 NVM Express SSD

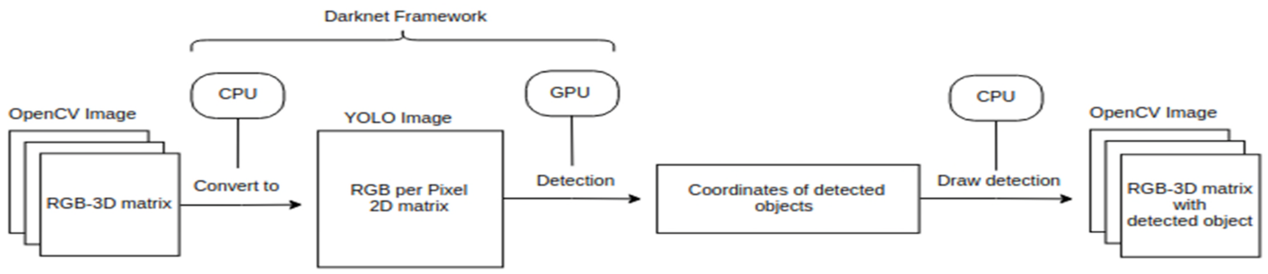

Figure 3. The architecture of the object detection process in MvcYOLO model.

B. Experimental Evaluation of the Model

As discussed before in this section, in order to evaluate the proposed model MvcYOLO, we make an experimental comparison between the original YOLOv3 and MvcYOLO. Our focus is on system resources like CPU, GPU, RAM and FPS values. CPU is one of the critical resources in this area because it performs the main conversion steps in the darknet framework. As previously mentioned, we prepare 10 videos in a different range of quality. The results are presented in Figure 4 in three-line graphs denoted as a, b, and c. The first line graph (a) compares the CPU usage percentage between original YOLOv3 and MvcYOLO in different video qualities. Overall, the CPU usage increases over the video qualities during the increasing computational power for handling large size of frames. The CPU usage in the MvcYOLO is lower than YOLOv3 overall video qualities. The bigger drop of CPU usage was seen on 4k video quality. We conclude that MvcYOLO has a better performance on high-quality videos.

The second line graph (b) uses the frame rate as a criterion for evaluation. The FPS is the frequency at which consecutive images, called frames, appear on a display. The FPS is compared in two models and the result shows the average of FPS values in the two approaches. It can be seen that the FPS of MvcYOLO was far higher than the original YOLOv3. The MvcYOLO improves FPS of high-quality video frames 13 to 26 in comparison to the original YOLOv3 approach. However, the MvcYOLO uses lower computational power in terms of CPU and RAM resources in each frame. As illustrated in the results, CPU usage has a high correlation with FPS. In 720p, the FPS value of YOLOv3 and MvcYOLO are 28, while this amount decreases steeply to 26 and continues to decline but more gradually to 13 in 4k video quality on YOLOv3 approach. In contrast, the FPS amounts of MvcYOLO declined more steeply to 26 in 1080p and 4k video qualities.

The third line graph (c) shows the average RAM usage in terms of GB over different video qualities in each approach. MvcYOLO uses less RAM than YOLOv3 because of doing less computation in the conversion process while improvements are not significant. Changes in RAM usage of YOLOv3 and MvcYOLO are approximately the same in each video quality.

C. Experimental Evaluation of the Multi-Thread Model

In the second part of the evaluation, our goal is to consider the proposed model in the previous section with a multi-thread approach as a solution to handle concurrent surveillance videos. As discussed in Section III, to obtain more efficiency in YOLO, we need to use a multi-thread architecture. So, we performed experiments to find the optimal number of cameras. Based on the result, 8GB GPU memory could handle a maximum of four surveillance cameras simultaneously. This paper provides a multi-thread approach to handle a minimum of four surveillance cameras over each YOLOv3 object and supports 16 cameras for 8GB GPU memory. The reason for choosing four cameras is the thread scheduler of the operating system and CPU power of the analysis server.

We call the multi-thread approach Multi-MvcYOLO. We compare MvcYOLO and Multi-MvcYOLO in three metrics of CPU usage, FPS and RAM usage. The results are depicted in Figure 5. Based on the results, it can be seen that CPU usage of Multi-MvcYOLO sharply decreased in each video quality especially on high-quality videos. In Multi-MvcYOLO the value of FPS is less compared to MvcYOLO because of performing four videos as multi-threads. However, Multi-MvcYOLO needs much more RAM than the others due to the thread scheduling process that uses extra RAM for scheduling.

As mentioned before, Multi-MvcYOLO handles four videos on a single YOLOv3 instance while YOLOv3 cannot handle more than one video on an object instance. Each YOLOv3 object instance uses approximately 1.70 GB from GPU memory. Since GPU memory is so costly, Multi-MvcYOLO approach uses optimal GPU memory and handles more surveillance cameras over a normal GPU card. Interestingly, Multi-MvcYOLO has better performance compared with YOLOv3 over the video qualities.

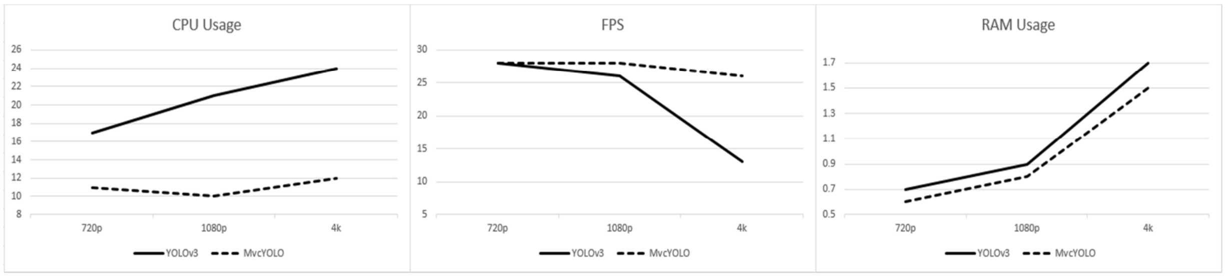

Figure 4. A comparison between original YOLOv3 and MvcYOLO in different video qualities on a) The CPU usage percentage b) FPS and c) RAM Usage

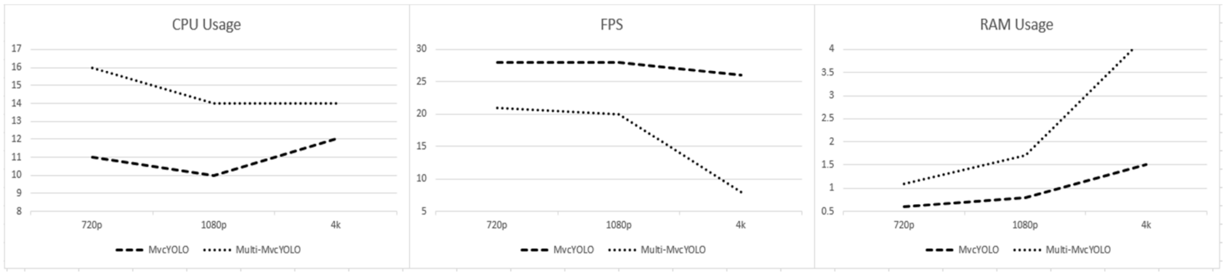

Figure 5. A comparison between original MvcYOLO and Multi-MvcYOLO in different video qualities on a) The CPU usage percentage b) FPS and c) RAM Usage

V. CONCLUSION AND FUTURE WORK

In the existing systems, it is necessary to use a powerful single GPU when applying YOLOv3 to detect objects. However, sometimes, there is a need to process multiple real-time object detection algorithms concurrently on a single GPU, where each object detection algorithm receives the live stream from a camera. In these cases, to reduce memory usage and other resources, we proposed solutions in two steps. In the first step, we provided a model for optimal memory usage and in the second step, we proposed a multi-thread approach that uses YOLOv3 to perform real-time object detection on multiple live streams concurrently on a GPU. This way, GPU resources are optimized to solve the limited memory issue. Experimental results show that the proposed approach can reduce memory consumption and increase performance by an average of 12% in CPU usage and 13% in FPS compared to the original YOLOv3. As future work, we are planning to improve MvcYOLO in terms of load balancing and network overhead on a large scale of camera surveillance environments.

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. I-647-I-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] 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.

[26] 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.

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

[28] B. Blanco-Filgueira, D. Garc´ıa-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.

[29] 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. 22782324, 1998.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.