





# Tensor-Based CUDA Optimization for ANN Inference Using Parallel Acceleration on Embedded GPU

Ahmed Khamis Abdullah Al Ghadani , Waleeja Mateen <sup>(✉)</sup> ,  
and Rameshkumar G. Ramaswamy

National University of Science and Technology, Muscat 111, Oman  
{ahmed150309, waleeja160359, rameshkumar}@nu.edu.om

**Abstract.** With image processing, robots acquired visual perception skills; enabling them to become autonomous. Since the emergence of Artificial Intelligence (AI), sophisticated tasks such as object identification have become possible through inferring Artificial Neural Networks (ANN). Be that as it may, Autonomous Mobile Robots (AMR) are Embedded Systems (ESs) with limited on-board resources. Thus, efficient techniques in ANN inferring are required for real-time performance. This paper presents the process of optimizing ANNs inferring using tensor-based optimization on embedded Graphical Processing Unit (GPU) with Computer Unified Device Architecture (CUDA) platform for parallel acceleration on ES. This research evaluates renowned network, namely, You-Only-Look-Once (YOLO), on NVIDIA Jetson TX2 System-On-Module (SOM). The findings of this paper display a significant improvement in inferring speed in terms of Frames-Per-Second (FPS) up to 3.5 times the non-optimized inferring speed. Furthermore, the current CUDA model and TensorRT optimization techniques are studied, comments are made on its implementation for inferring, and improvements are proposed based on the results acquired. These findings will contribute to ES developers and industries will benefit from real-time performance inferring for AMR automation solutions.

**Keywords:** Artificial Neural Networks · Embedded GPU · TensorRT · Real-time · NVIDIA Jetson · Image processing · YOLO · CUDA

## 1 Introduction

Artificial Intelligence (AI) is being increasingly adopted into ES due to its synergistic relation in the virtue of efficient data analysis, capability to rationalize tasks and performing system optimization. Over the past few years, AI has been rapidly developed to be utilized in applications such as safe autonomous driving which involves object avoidance and collision mitigation [1] and in visual perception tasks like scene understanding, object detection, and localization [2, 3]. Machine Learning, a prime constituent of AI comprises of two main processes: training and inferring. These are power-intensive and computationally demanding tasks [4].

© IFIP International Federation for Information Processing 2020

Published by Springer Nature Switzerland AG 2020

I. Maglogiannis et al. (Eds.): AIAI 2020, IFIP AICT 583, pp. 291–302, 2020.

[https://doi.org/10.1007/978-3-030-49161-1\\_25](https://doi.org/10.1007/978-3-030-49161-1_25)

As observed in examples [5, 6], inferencing is a critical step in implementing and deploying an effective ES. It encompasses the procedures involved in determining a viable conclusion from the evidence collected during the training phase. Observing from the perspective of ESs, inferencing which is a power-intensive utility, is challenging to run with limited available resources [7]. In such a scenario, optimization is the key to achieving best performance with lower power consumption [8].

System optimization is a challenging step in achieving higher performance and therefore, the relatively more popular alternative is usually opted for: scaling the system over several GPUs [9]. This solution incurs high cost and decreases projects' economic feasibility. The research [10] supports this observation through a study conducted on ImageNet competition which aims to achieve maximum accuracy through complex Deep Neural Networks (DNNs). To maximize performance, the participants neglect system optimization and invest in hardware upgrades, which shows inefficient resources utilization as the existing system was not executed up to its full potential.

Optimization encompasses processes ranging from data-handling, parameters fine-tuning, architecture selection and process pipeline remodeling [11]. This can be seen in examples like [12] open source GPU-Accelerated feature extraction tool and Bonnet framework for semantic segmentation in robotics [13]. This is also evident in the emergence of GPU acceleration frameworks like CUDA and OpenGL [8]. Another key technology in optimization is TensorRT [14]. Yamane [15] stressed on the importance of inferencing on ESs. Also, highlighting the fact that ESs are still lacking in real-time performance, rendering edge devices unreliable for critical missions.

In this paper, You-Only-Look-Once (YOLOv3) object detection system is inferred on Jetson TX2. YOLO model predicts objects by running a single network evaluation [16]. Making it lighter than most networks with similar accuracy; hence, ideal candidate for ESs. Running on the GPU for parallelization, the inference performance is evaluated with and without optimization using TensorRT inference accelerator. The acquired results present an eye-catching increase in performance up to 3.5x times. This provides an insight into optimization trends and their effectiveness. Furthermore, with the acquired insights, CUDA framework pipeline is studied under limited scope and areas of potential improvements are suggested.

## 2 Methods

This research is a part of a bigger research that focuses on applying visual Simultaneous Localization And Mapping (vSLAM) for AMRs in industrial environments. With focus on material handling in warehouses; this work has been presented in the Logistic Research Competition organized by Oman Logistic Center. Additionally, for ore mining industry, a dataset with one class "Power Screens" has been initiated [17]. Both of which targeted real-time inferencing of YOLOv3 ANN. This research continues the previous work by optimizing the inferencing speed to meet real-time needs with a pretrained YOLOv3 ANN with COCO dataset with 80 classes.

### 2.1 Hardware Setup

The hardware setup follows certain specifics for AMRs: limited on-board compute capability, limited power source, System-On-Module (SOM). The physical set-up of the robot adopts the open source Turtlebot3, with adjustments for Jetson TX2, ZED Stereo Camera, 2D RP-LIDAR, and a Logitech C920 HD Pro webcam. On the execution layer, as can be seen on the AMR system block diagram in Fig. 1, the robot uses differential drive for movements with magnetic encoders for odometry. Be that as it may, in this AMR context, this research focuses solely on the optimization process of ANN inferencing.

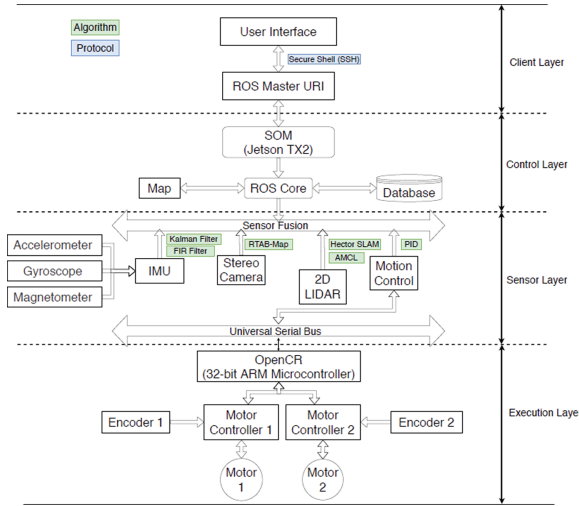


Fig. 1. AMR system block diagram

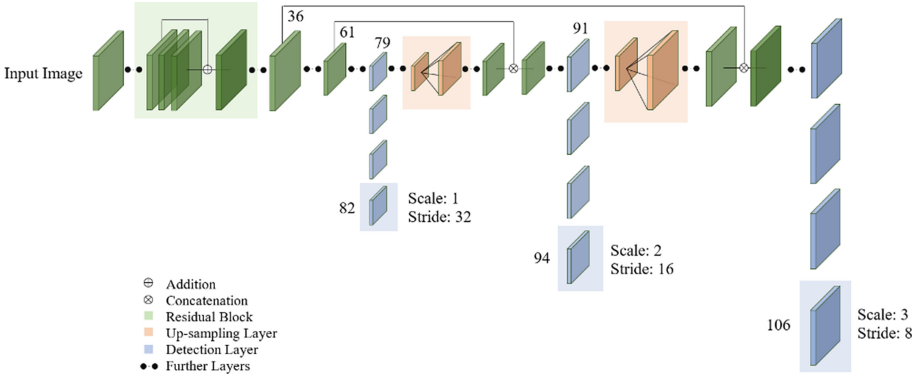
### 2.2 YOLO ANN for Object Detection

YOLO ANN implements a smart approach for object detection. Achieving real-time performance using a single Convolutional Neural Network (CNN) structure. Quoting its developers from their research paper [18] “It achieves 57.9 AP50 in 51 ms on a Titan X, compared to 57.5 AP50 in 198 ms by RetinaNet, similar performance but 3.8× faster”.

YOLO ANN architecture is based on an open-source CNN called Darknet-53 for image classification. Darknet-53 serves as the network’s backbone with an additional 53 layers for detection. Resulting in a total of 106 layers, out of which 75 are convolutional, 23 shortcuts, 4 routes, 2 upsamples, 3 detections [19].

With reference to the architecture illustration in Fig. 2, the network performs detection at 3 different scales at 82nd, 94th, and 106th layers. At each one of the detection layers, the image is downsampled by a factor called stride of the network. Respective to the detection layers, the input image is downsampled by a factor of 32, 16, and 8.

In this architecture, the detection kernel shape is defined by  $N \times N \times (B \times (5 + C))$ .  $N \times N$  being the alternating  $1 \times 1$  convolutional layers for feature space reduction.



**Fig. 2.** YOLO network architecture

$B = 3$  being the number of anchors, resulting in the prediction of three bounding boxes per cell. These anchors are priors for bounding boxes that were calculated on the COCO dataset using k-means clustering. This dataset has 80 classes; hence  $C = 80$ . The ‘5’ is the sum of the 4 bounding box offsets from the cluster centroids, using the logistic sigmoid Eq. (1) and objectness prediction score 1. Thus, the detection kernel was designed as  $1 \times 1 \times (3 \times (5 + 80))$  tensor or its equivalent  $1 \times 1 \times 255$  tensor for each scale. For the bounding box predictions,  $t_x, t_y, t_w, t_h$  determine 4 co-ordinates of each bounding box,  $(c_x, c_y)$  are co-ordinates at the offset from top corner of the detection grid.  $p_w, p_h$  are width and height of bounding box priors. While  $b_x, b_y$  are  $x, y$  centre co-ordinates of prediction and  $b_w, b_h$  are width and height of the predicted bounding box.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

$$b_x = \sigma(t_x) + c_x \tag{2}$$

$$b_y = \sigma(t_y) + c_y \tag{3}$$

$$b_w = p_w e^{t_w} \tag{4}$$

$$b_h = p_h e^{t_h} \tag{5}$$

Considering the above, the output value is always between 0 and 1 due to the logistic sigmoid function. YOLO ANN predicts the relative offsets of the bounding box center rather than the absolute coordinates. Normalized by the feature map dimensions which is 1 [20].

### 2.3 Complexity Class

To verify the complexity class in which the adopted YOLO ANN falls in; it is necessary to examine the decision-making procedure in determining a prediction. Bearing that in

mind, it is worth noting that “object recognition is not a formally defined problem, so is not in itself either polynomial time solvable or NP-complete” [21].

YOLO ANN recognizes an object based on class-specific confidence score for each bounding box. This is achieved by first computing the confidence as follows:

$$\text{Confidence} = \text{Pr}(\text{Object}) \times \text{IOU}_{\text{pre}}^{\text{truth}} \quad (6)$$

The value of  $\text{Pr}(\text{Object})$  is 1 if a ground truth box is present within the grid cell, otherwise it is 0. While the intersection over union value between the predicted bounding box and the actual ground truth is denoted by  $\text{IOU}_{\text{pre}}^{\text{truth}}$  [22].

With reference to the detection kernel in Sect. 2.2, each grid generates conditional class probabilities represented as follow:  $\text{Pr}(\text{Class}_i | \text{Object})$ . YOLO ANN output is class-specific confidence, computed by multiplying the conditional class probabilities and the individual box confidence predictions as represented by [22]:

$$\text{Pr}(\text{Class}_i | \text{Object}) \times \text{Pr}(\text{Object}) \times \text{IOU}_{\text{pre}}^{\text{truth}} = \text{Pr}(\text{Class}_i) \times \text{IOU}_{\text{pre}}^{\text{truth}} \quad (7)$$

## 2.4 Tensor-Based Optimization

YOLO ANN was originally developed using Darknet framework written in C and CUDA [16]. Optimization using TensorRT adds an extra step between training a model and inferencing it. That step requires the trained ANN model to be converted into a format that is optimizable by TensorRT. TensorRT is a programmable inference accelerator built on CUDA for parallel programming. In this research, the conversion of the YOLO ANN model is done through an open-source format defined by the Open Neural Network Exchange (ONNX) ecosystem. This format ensures interoperability and easy hardware access optimization.

When approaching an optimization problem on an AMR system, specific considerations pertaining to ESs must be put into account. Memory management, hardware architecture utilization, inference precision, and power efficiency. TensorRT memory allocation is done using TensorFlow allocators by setting argument `config.gpu_options.per_process_gpu_memory_fraction` [23]. For hardware architecture utilization, ANN model is converted on target Jetson TX2.

Inference in INT8 and mixed precision reduces memory footprint; which is important on an ES. Using symmetric linear quantization, models running in 32-bit floating-point precision are scaled down to 8 bits with preserved symmetry. FP32 represents billions of numbers while the INT8 represents 256 possible values only [24]. Equation below formulates this process of getting quantized INT8 value, where input, floating point range, and scaling factor are denoted by  $x$ ,  $r$  and  $s$  respectively:

$$\text{Quantize}(x, r) = \text{round}(s * \text{clip}(x, -r, r)) \quad (8)$$

Using calibration and quantization aware training, accuracy is preserved when model is scaled to INT8. Taking a specific range where most of the activation values fall.

The network’s performance is measured through latency and throughput. The former is determined by the time elapsed between input presence until output is

acquired. The latter is determined by the number of inferences performed in a set amount of time. “Both can be measured using high precision timers present in C++ `std::chrono::high_resolution_clock`, and monitored by profiling CUDA and memory utilization during runtime [25].”

### 3 Experimental Procedure

The experiment was conducted on the following setup with the webcam mounted on an AMR. The robot was set to move at a constant speed of 10 cm/s detecting objects set in its field of view. The objects detected are apparent on the screen attached (Fig. 3).



**Fig. 3.** Objects detected displayed in real-time

The process of optimizing an ANN, especially for inferencing on an ES; requires a deep understanding of the software and the hardware architecture. This research is carried on Jetson TX2 SOM running Linux for Tegra (L4T). The key technical specification of this system can be seen in the Table 1 taken from [26].

**Table 1.** Jetson TX2 module specifications

GPU	256-core NVIDIA Pascal™ GPU architecture with 256 NVIDIA CUDA cores		
CPU	Dual-Core NVIDIA Denver 2 64-Bit CPU Quad-Core ARM® Cortex®-A57 MPCore		
Memory	8 GB 128-bit LPDDR4 Memory 1866 MHx - 59.7 GB/s		
Storage	32 GB eMMC 5.1	Power	7.5 W/15 W

It is noted that the Jetson board is running a 256-core Nvidia GPU which is based on Pascal architecture [26]. An efficient optimisation technique for GPU parallelization should be able to optimally distribute the workload on the 256 cores available.

Since YOLO was originally developed using Darknet framework, optimizing this ANN will require adjustments starting from the framework. Hence, the framework was built from source on the Jetson TX2 with CUDA, cuDNN, and OpenCV enabled. This is done by setting their respective flags to 1 in the Makefile. Adding to that, it is important to specify the CUDA architecture, which is 62, for Jetson TX2.

On the time of conducting this experiment, Jetson TX2 was running L4T, CUDA, and other relevant libraries as seen in Table 2 below:

**Table 2.** Software components and their versions on Jetson TX2

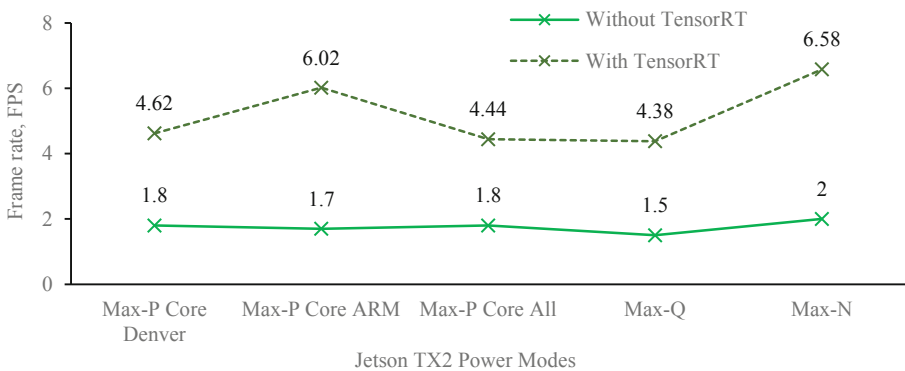
Component	Version	Component	Version
JetPack	4.2.1	cuDNN	7.5.0.66
OS	L4T R32.2 (K4.9)	TensorRT	5.1.6.1
CUDA	10.0.326	OpenCV	3.4.0

At first YOLOv3-416 was run using a direct implementation of Darknet detector as documented by Redmon & Farhadi [27]. Even though Darknet was built with CUDA and cuDNN enabled, the GPU utilization was low. This resulted in poor performance; maxing at 2 FPS. For efficient GPU utilization, YOLO ANN is formatted according to ONNX open format. Facilitating the conversion to TensorRT using PyCUDA API to access parallel computation of the GPU.

## 4 Result and Discussion

### 4.1 System Performance

Obtained results were analyzed for comparison between FPS with and without TensorRT optimization at different power modes as plotted in Fig. 4:



**Fig. 4.** FPS performance analysis with and without TensorRT optimization

As inferred from Fig. 4, when YOLOv3 is executed on CUDA along with OpenCV, the maximum achieved FPS was 2.0 at Max-N. This power mode offers maximum GPU frequency and utilizes both, ARM A57 and Denver cores. The lowest FPS was obtained from the Max-Q power mode at 1.5 FPS and was also observed to drop to 1.3 FPS when input traffic for object detection was increased. Max-Q and Max-P Core All modes had fluctuations in output. However, Max-P Core ARM was observed to be the most stable as the FPS stayed constant at 1.7. Max-N and Max-P Core Denver were observed to have minimum drop of  $-0.1$  FPS.

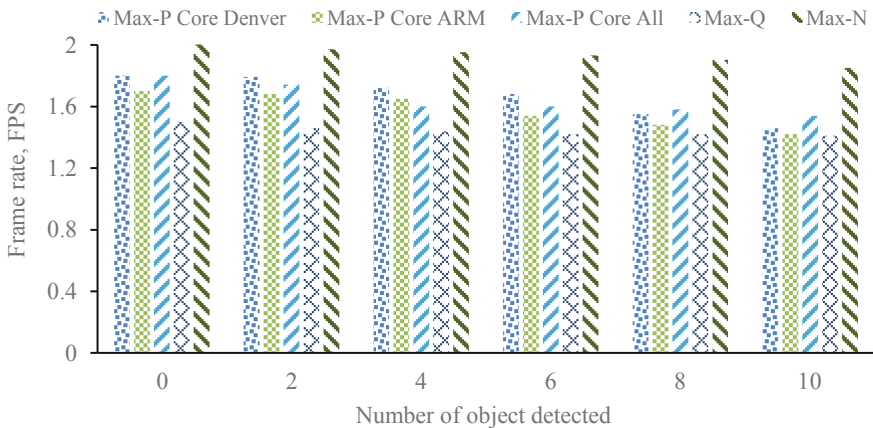
In contrast to the above, with TensorRT optimization, the performance showed a drastic improvement from 147% in Max-P Core All to 254% improvement in Max-P Core ARM. The highest FPS achieved was on the Max-N at 6.58 units; an increase of  $3.5\times$  in performance. This is important for object detection applications on ESs from a live stream. Table 3 summarizes the performance gain with and without TensorRT.

**Table 3.** Percentage increase in FPS obtained without versus with TensorRT

Power modes	Without TensorRT	With TensorRT	Percentage increase, %
Max-P Core Denver	1.8	4.62	157
Max-P Core ARM	1.7	6.02	254
Max-P Core All	1.8	4.44	147
Max-Q	1.5	4.38	192
Max-N	2.0	6.58	229

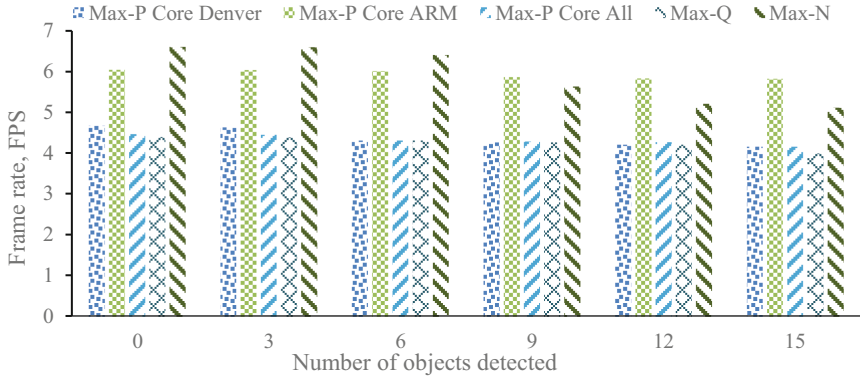
### 4.2 Precise Object Detection

Adding one more dimension to the comparison, the effect of FPS on the number of objects detected is observed. Performance in FPS and detection rate were monitored on a test scene including 15 objects belonging to the ANN classes. As observed in Fig. 5, without optimization, maximum 10 objects were detected out of 15.



**Fig. 5.** Frame rate versus number of objects detected without TensorRT optimization





**Fig. 6.** Frame rate versus number of objects detected with TensorRT optimization

When tested with TensorRT optimization, all 15 objects in the scene were detected while the FPS was relatively higher than without optimization as shown in Fig. 6.

Both Fig. 5 and Fig. 6 illustrate the relation between FPS, number of objects detected, and power modes. It can be deduced that the increase in FPS results in more objects detected. However, more processing power is required as more objects must be processed per frame. The system allocates more resources towards recognition of objects present in the frame. This results in drop in FPS, consequently lowering the accuracy as Table 4 and Table 5 clarify. Additionally, 5 objects failed to be detected without optimization as highlighted in Table 4.

**Table 4.** Observation of objects detected without TensorRT and their percentage accuracy

#	Detected	Accuracy	#	Detected	Accuracy	#	Failed to detect
1	Bottle	100%	6	Keyboard	91%	11	Mouse
2	Bottle	93%	7	Backpack	88%	12	Laptop
3	Person	100%	8	Cell phone	87%	13	Keyboard
4	Person	99%	9	Cell phone	84%	14	Clock
5	Keyboard	99%	10	Cup	99%	15	Table

**Table 5.** Observation of objects detected with TensorRT and their percentage accuracy

#	Objects detected	Accuracy	#	Objects detected	Accuracy
1	Bottle	100%	9	Cell phone	96%
2	Bottle	92%	10	Cup	100%
3	Person	97%	11	Backpack	91%
4	Person	89%	12	Mouse	37%

(continued)

**Table 5.** (continued)

#	Objects detected	Accuracy	#	Objects detected	Accuracy
5	Keyboard	99%	13	Laptop	99%
6	Keyboard	67%	14	Table	73%
7	Keyboard	67%	15	Clock	99%
8	Cell phone	96%			

### 4.3 CUDA Study

CUDA framework is arguably the most efficient General-Purpose GPU (GPGPU) platform. This is mainly due to its maturity, continuous update and support from NVIDIA [28]. However, to accommodate the ever-increasing range of GPUs and their architectures, NVIDIA moved into generalization and multi-layering of CUDA 5.0. This caused a noticeable reduction in performance when compared to CUDA 4.0 [29]. Adding to that, it was noted in [30] that CUDA GPU acceleration was performing best on certain workloads and for full utilisation, GPU kernels must be optimised as well.

## 5 Conclusion and Future Work

Optimization is the key in achieving maximum efficiency of any given ES running on AI. This paper identifies the causes of low accuracy and decreased performance of inferencing ANNs on ESs. Using Jetson TX2 SOM, a comparison was made between YOLOv3 ANN running on CUDA with and without tensor-based optimization using TensorRT inference accelerator. Making use of open-source format (ONNX) for conversion from YOLO native format to TensorRT.

Inferencing performance and accuracy are seen to be linked. As seen in Sect. 4.2, high FPS allow for accurate perception of the current scene. Enabling AMRs to detect more objects faster, resulting in prompt response. Nevertheless, complex scenes pose computational challenges that may affect the resource allocation, lowering the FPS.

The optimization technique focuses on reducing the memory footprint and computational demands. Accomplished through downscaling the ANN model from FP32 to INT8 while targeting intermediate activation layers. Although this increases the performance in FPS significantly by  $3.5\times$  and number of objects detected but reduces the detection accuracy. This analysis conjoins the aforementioned parameters with power efficiency, pertaining to AMR specifics.

This comprehensive study gives a thorough insight towards optimization of ESs for effective resources utilization. Future works include in-depth analysis of the processes in terms of mean Average Precision (mAP) and studying the relationships between the several elements of the existing frameworks and APIs using a visualization software such as NVIDIA Nsight Systems.

**Acknowledgment.** This research has been funded conjointly by Oman Logistics Center in Asyad Group and the National University under NU Grant Call 2019. Special appreciation to Mr. D. Ragavesh for his help with image processing. Additionally, sincere gratitude to Mr. M. J. Varghese

for his help with Artificial Neural Networks and guidance in performance evaluation. Lastly, special thanks to J. K. Jung for sharing his experience on tensor optimization for Embedded Systems; which was a prominent guide in the making of this research.

## References

1. Okuyama, T., Gonslaves, T., Upadhay, J.: Autonomous driving system based on deep Q learning. In: 2018 International Conference on Intelligent Autonomous Systems (ICoIAS), pp. 201–205. IEEE; Singapore (2018). <https://doi.org/10.1109/icoias.2018.8494053>
2. Feng, X., Jiang, Y., Yang, X., Du, M., Li, X.: Computer vision algorithms and hardware implementations: a survey. *Integration* **69**, 309–320 (2019). <https://doi.org/10.1016/j.vlsi.2019.07.005>
3. Ray, J., Thompson, B., Shen, W.: Comparing a high and low-level deep neural network implementation for automatic speech recognition. In: Kellenberger, P. (ed.) *First Workshop for High Performance Technical Computing in Dynamic Languages*, pp. 41–46. IEEE Service Center, Piscataway (2016). <https://doi.org/10.1109/HPTCDL.2014.12>
4. Rungsuptaweekoon, K., Visoottiviseeth, V., Takana, R.: Evaluating the power efficiency of deep learning inference on embedded GPU systems. In: 2017 2nd International Conference on Information Technology (INCIT), pp. 1–5. IEEE, Nakhon Pathom (2017). <https://doi.org/10.1109/incit.2017.8257866>
5. Marco, V.S., Taylor, B., Wang, Z., Elkhatib, Y.: Optimizing deep learning inference on embedded systems through adaptive model selection. *ACM Trans. Embed. Comput. Syst.* **19**(1), 1–28 (2020). <https://doi.org/10.1145/3371154>
6. Yoo, S., et al.: Structure of deep learning inference engines for embedded systems. In: *The 10th International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju Island, South Korea, pp. 920–922 (2019). <https://doi.org/10.1109/ictc46691.2019.8939843>
7. Zhou, X., Wu, P., Zhang, H., Gou, W., Liu, Y.: Learn to navigate: cooperative path planning for unmanned surface vehicles using deep reinforcement learning. *IEEE Access* **7**, 165262–165278 (2019). <https://doi.org/10.1109/access.2019.2953326>
8. Aldegheri, S., Manzato, S., Bombieri, N.: Enhancing performance of computer vision applications on low-power embedded systems through heterogeneous parallel programming. In: *Proceedings of the IFIP/IEEE International Conference on Very Large-Scale Integration (VLSI-SoC)*, Verona, Italy, pp. 119–124 (2018). <https://doi.org/10.1109/vlsi-soc.2018.8644993>
9. Saxerud, A.L., Ferrell, J.P., Dunn, E.A.: Application of the CUDA® toolkit multi-GPU libraries to an out-of-core MoM solver. In: *Proceedings of the 2016 IEEE Antennas and Propagation Society International Symposium (APSURSI)*, Fajardo, Puerto Rico, pp. 2013–2014 (2016). <https://doi.org/10.1109/aps.2016.7696713>
10. Canziani, A., Culurciello, E., Paszke, A.: Evaluation of neural network architectures for embedded systems. In: 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, pp. 1–4 (2017). <https://doi.org/10.1109/iscas.2017.8050276>
11. Kodra, A.: *Machine Learning Model Optimization* (2019). <https://heartbeat.fritz.ai/machine-learning-model-optimization-9fbcf9f6990>. Accessed 20 Feb 2020
12. Michálek, J., Vaněk, J.: An open-source GPU-accelerated feature extraction tool. In: Yuan, B., Ruan, Q., Tang, X. (eds.) *2014 IEEE 12th International Conference on Signal Processing Proceedings*, HangZhou, China, pp. 450–454 (2019). <https://doi.org/10.1109/iscas.2017.8050276>

13. Milioto, A., Stachniss, C.: BonNet: an open-source training and deployment framework for semantic segmentation in robotics using CNNs. In: 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, pp. 7094–7100 (2019). <https://doi.org/10.1109/icra.2019.8793510>
14. Xu, R., Han, F., Ta, Q.: Deep learning at Scale on NVIDIA V100 accelerators. In: Proceedings of the 2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High-Performance Computer Systems (PBMS), Dallas, TX, USA, pp. 23–32 (2018). <https://doi.org/10.1109/pbms.2018.8641600>
15. Yamane, S.: Deductively verifying embedded software in the era of artificial intelligence = machine learning + software science., In: 2017 IEEE 6th Global Conference on Consumer Electronics (GCCE), Nagoya, Japan, pp. 1–4 (2017). <https://doi.org/10.1109/gcce.2017.8229475>
16. Redmon, J., Farhadi, A.: YOLO: real-time object detection (2010). <https://pjreddie.com/darknet/yolo/>. Accessed: 19 Jan 2020
17. Al Ghadani, A.K.A., Ramaswamy, R.G.: Collecting datasets on ore mining industry and training using YOLO ANNs for object identification in mining sites. *J. Big Data Smart City* **1**, 1–6 (2019). MEC, Muscat, Oman
18. Redmon, J., Divvala, Girshick, Farhadi A.: YOLO9000: better, faster, stronger. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, pp. 6517–6525 (2016). <https://doi.org/10.1109/cvpr.2017.690>
19. GitHub: Darknet/YOLOv3.config at master – PJReddie/Darknet (2018). <https://gist.github.com/fabito/a49bb6a5593594f26275bc90baba6e32>. Accessed 20 January 2020
20. Smith, S.W.: Neural network architecture (1999). <https://www.dspguide.com/ch26/2.htm>. Accessed 18 Jan 2020
21. Impagliazzo, R.: A personal view of average case complexity. In: Proceedings of Structure and Complexity Theory, Minneapolis, MN, USA, pp. 134–147 (1995). <https://doi.org/10.1109/sct.1995.514853>
22. Wu, J.: Complexity and accuracy analysis of common artificial neural networks on pedestrian detection. In: 2018 2nd International Conference on Electronic Information Technology and Computer Engineering (EITCE), Shanghai, China (2018). <https://doi.org/10.1051/mateconf/201823201003>
23. NVIDIA: Deep learning frameworks documentation (2020). <https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html>. Accessed 1 Feb 2020
24. Davoodi, P., Lai, G., Morris, T., Sharma, S.: High performance inference with TensorRT Integration (2019). <https://blog.tensorflow.org/2019/06/high-performance-inference-with-TensorRT.html>. Accessed 2 Feb 2020
25. NVIDIA: TensorRT documentation (2020). <https://docs.nvidia.com/deeplearning/tensorrt/best-practices/index.html>. Accessed 2 Feb 2020
26. NVIDIA: Jetson TX2 module (2020). <https://developer.nvidia.com/embedded/jetson-tx2>. Accessed 19 Jan 2020
27. Redmon, J., Farhadi, A.: YOLOv3: an incremental improvement (2018). <https://arxiv.org/pdf/1804.02767.pdf>. Accessed 3 Feb 2020
28. Cook, S.: CUDA Programming: A Developer's Guide to Parallel Computing with GPUs, pp. 13–19. Elsevier Science & Technology, Saint Louis (2012)
29. Wezowicz, M., Tauber, M.: On the cost of a general GPU framework – The strange case of CUDA 4.0 vs. CUDA 5.0. In: SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA, pp. 1535–1536 (2012). <https://doi.org/10.1109/sc.companion.2012.310>
30. Hwu, W.M., Rodrigues, C., Ryoo, S., Stratton, J.: Compute unified device architecture application suitability. *Comput. Sci. Eng.* **11**(3), 16–26 (2009). <https://doi.org/10.1109/MCSE.2009.48>