Integration of Deep Learning Computer Vision Algorithms in PXIe Ecosystems

Sanjib Sarkar
Acquired Data Solutions, Inc
Rockville, MD USA
sanjib.sarkar@acquiredata.com
ORCID: 0000-0003-2574-8392

Mandrita Banerjee
Department of Electrical & Computer Engineering
Florida International University
Miami, FL USA
mbanerje@fiu.edu
ORCID:0000-0003-2712-4735

Ross Q. Smith
RADX Technologies, Inc
Palo Alto, CA, USA
rossqsmith@radxtech.com

Steven Seiden
Acquired Data Solutions, Inc
Rockville, MD USA
steve.seiden@acquiredata.com

***Abstract*—The PXIe (PCI eXtensions for Instrumentation) based software-defined instruments are widely employed in manufacturing, quality assurance (QA), and repair-related test and measurement (T&M) applications for their superior modularity, consistent performance, and outstanding life-cycle support. Such applications often require components and subsystems for visual or sensor-based inspection, quality control, detection, and analysis for a wide variety of purposes. Traditional rule-based systems for visual inspection, detection, and analysis require human-engineered feature extraction. Thus, the accuracy and reliability of these traditional systems directly depend on the extracted features and the methods used for feature extraction. In contrast, deep learning algorithms such as Convolutional Neural Networks (CNNs) extract and learn features autonomously from large datasets. The "You Only Look Once" (YOLO) algorithm is a prominent deep learning-based method used in many computer vision applications due to its highly efficient feature learning capabilities compared to traditional vision algorithms. YOLO is particularly notable for its ability to perform real-time object detection with impressive speed and accuracy, making it suitable for dynamic environments where rapid decision-making is critical. In this article, we present the integration of a YOLO V5 model in a PXIe system equipped with a RADX PXIe-GPU and RADX PXIe-SSD storage module to rapidly and accurately detect and count mechanical objects. We achieved more than 90% accuracy of the trained model to detect mechanical objects on a setup that simulates a high-throughput production line. The integration of the YOLO V5 model within the PXIe ecosystem demonstrates a significant advancement in the field of automated inspection and analysis on this important platform. Doing so not only enhances the efficiency of the inspection process, but also reduces the likelihood of errors associated with manual inspection. This flexibility is crucial for industries that require continual innovation and improvement in their T&M processes..**

***Index Terms*—Artificial Intelligence (AI), Deep Learning (DL), You Only Look Once (YOLO), Object detections, Computer vision systems, PXIe, Catalyst-GPU.**

I. Introduction

Rapid developments in artificial intelligence (AI), including AI-based image processing algorithms, offer to greatly improve manufacturing, Q/A, and the repair-related test and measurement capabilities. The PXIe (PCI eXtensions for Instrumentation) [1], [2] is a PC-based modular and rugged platform for benchtop and automated test and measurement systems. Due to their repeatable and reproduceable nature, integrated features, long life cycle and high performance, PXIe systems are preferred by engineers and organizations including the USAF, the US Navy, the US Marine Core and the US Army for a wide range of mission critical T&M applications, including R&D, production, QA and repair-related activities in dozens of performance-centric industries including aerospace and defense, machine monitoring, automotive, medical devices, consumer electronics and many more. Many of the above applications require visual and/or sensor based inspection, detection, and analysis for a wide variety of purposes, including anomaly detection, defect detection, dimensional analysis, safety and security monitoring, package and label detection, assembly defect detection, quality monitoring, and many more. Traditional computer vision (CV) algorithms employ rule-based comparisons and convolutions with reference images to de- termine compliance or detect defects. Traditional CV requires human-engineered feature selection and algorithms to extract features from images to identify similarities. Algorithms like SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features), and BRIEF (Binary Robust Independent Elementary Features) are examples of conventional methods used to extract the desired features from raw images [3], [4]. However, these traditional CV methodsare inflexible and often require significant precision in fixturing to ensure valid data acquisition and comparisons. Unlike rule-based CV approaches, AI-based approaches use compute-intensive neural-network (or similar) approaches that "train" the system to recognize features in images or videos based on perceived similarities to objects in the training database that have been labelled by humans or AI-based labelling software. With AI, the confidence of an algorithm is largely dependent on the AI model's dataset size, and the degree of training employed. Prior to the ad-

vent of NVIDIA GPUs and their ability to perform literally trillions of operations per second, industrial AI applications were uncommon due to the prohibitive computational intensity required to perform model training and inference, especially for embedded or factory floor systems. With the advent of powerful GPUs, combined with their ease-of-programming and low cost, DL- based computer vision systems have become widely available in a variety of markets. The two commonly used AI-based object detection algorithms are divided into two categories: one-stage algorithms and two-stage algorithms [5]. In one stage algorithms, object localization and classification occurs concurrently whereas a two-stage algorithm breaks the object localization and classification into two separate stages. "You Only Look Once" (YOLO) [5], "Single-Shot Detector" (SSD) and RetinaNet fall into the one-stage algorithm category. Region-based Convolutional Neural Network (R-CNN), Fast RCNN, Mask R-CNN etc., are two-stage algorithms. Among these algorithms, YOLO is the most popular due to its combination of speed, useability, accuracy and flexibility. The three main advantages of using YOLO are as follows:

1) Speed - because YOLO frames detection as a regression problem, thereby eliminating the requirement of a complex pipeline, which in turn significantly reduces processing time per frame.
2) Context - YOLO implicitly encodes contextual information about classes as well as their appearance because it sees the entire image during training and test time.
3) Flexibility - YOLO possesses a generalized learning capability that allows it to account for environmental changes (such as in lighting or camera angle) and still accurately produce inferences.

YOLO's advantages are what led us to implement it in our vision systems. Additional details and architecture of the YOLO algorithm can be found in the reference [5]. In this article, we will cover our efforts and results in integrating deep learning-based object detection algorithms with open-source python packages, operating on a RADX Trifecta PXIe GPU deployed in a PXIe system [6].

The rest of the article is organized as follows: Section II discusses GPUs. In Section III, provides an overview of a PXIe system. We discuss the experimental setup and data collection in Section IV. In Section V, we describe data preprocessing and training the algorithm; and Section VI is about results and discussion. Finally, Section VII presents conclusive remarks.

II. GRAPHICAL PROCESSING UNIT (GPU)

The GPU, which was initially designed for transforming, rendering, and accelerating perspective-correct 3D graphics, was introduced in the mid to late 1990s by 3Dfx, NVIDIA and others [7], [8]. The architectural design between a CPU and a GPU differs in latency and throughput. A CPU is a serial processor based on a Von Neuman architecture that executes instructions sequentially and is designed to minimize the latency of executing instructions, whereas a GPU is a Single Instruction Multiple Thread (SIMT) stream processor that simultaneously executes a function (kernel) on a set of input data (stream), and is designed to maximize throughput. Fig. 1a and 1b show the canonical architecture of a CPU and a GPU, respectively. As indicated, the size of the cache memory area in the CPU is larger than that of a GPU. This architecture reduces access latency, but also limits the memory bandwidth. Whereas, to execute a massive number

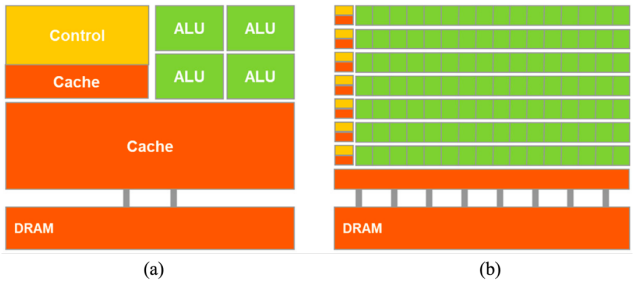

Fig. 1. The design architecture of (a) a CPU and (b) a GPU; the area of ALU is bigger in GPU compared to CPU while cache size is less in GPU compared to CPU [7][8].

of threads and parallel operations, the GPU is designed with minimized control logic and has smaller memory caches so that more chip area can be dedicated to Arithmetic and Logic unit (ALU) for performing floating point arithmetic. A more detailed discussion on GPUs can be found in the references [7], [8].

III. PXI SYSTEM

A PXIe system is a PC-based platform which provides power, cooling, timing and synchronization and a PCIe-based communication bus to support multiple instrumentation and computing modules with the same enclosure. PXIe system are used for benchtop and automated T&M applications. PXIe systems include three main hardware components – chassis, controller, and peripheral modules. The controller unit can be either a remote-control module, enabling the PXI system to be controlled from desktop, laptop, workstation, or server; or it can be a high-performance embedded controller with either a Microsoft Windows, Linux or other OS. Fig.2 shows an NI PXIe system with an NI PXIe-1095 Chassis, an NI PXIe-8880 embedded controller and several measurement or peripheral modules [1], [2].

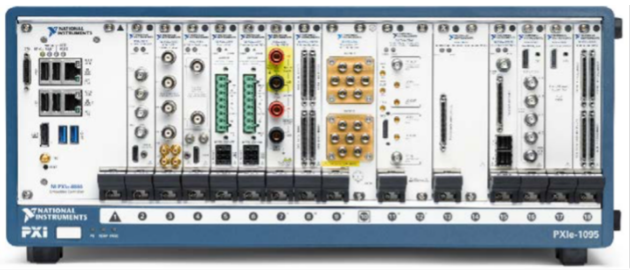

Fig. 2. A PXIe system - comprises of chassis, embedded controller, and measurement modules [1][2].

In 2022, RADX Technologies, Inc. introduced the first PXIe-GPUs, based on NVIDIA Small Form Factor PCIe GPUs and Embedded / Mobile GPUs shown in figure 3.

Fig. 3. RADX NVIDIA-based Catalyst and Trifecta PXIe-GPUs

IV. SETUP AND DATA COLLECTION

The setup, which is shown in Fig. 4, is consists of a PXIe-1095 chassis with NI PXIe - 8880 Embedded Controller (CPU with 2.3 GHz, 8C/16T Intel Xeon E5-2618L v3), RADX Trifecta PXIe-GPU with an NVIDIA RTX A2000 Embedded GPU, a 12 megapixel USB-connected camera system with 720P resolution, an ring illuminator with variable light temperature and intensity, and a turntable, which simulates a moving assembly line throughput, simulate a variable-throughput production environment via variable rotational speed. The camera is mounted above the turntable, which features three types of industrial objects: washers, nuts, and screws, scattered randomly across its surface. To train the system, we collected over 500 images with various orientations and environmental conditions such as illuminator at minimum and maximum intensity, camera slightly out of focus, different counts of item etc.). Images were collected and stored on a high-performance RADX Trifecta PXIe SSD module.

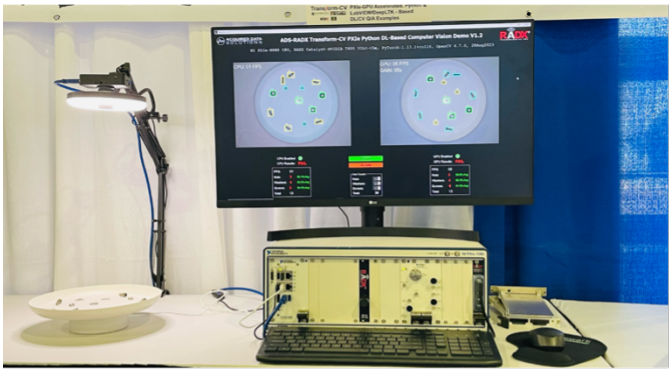

Fig. 4. PXIe Computer Vision setup consisting of a PXIe system with chassis, embedded controller with USB interface, PXIe-SSD and PXIe-GPU, a camera, an illuminator, and a turntable, which simulates a conveyor belt in manufacturing and assembly production facilities.

V. DATA PREPROCESSING AND TRAINING

After collecting the images, we annotated the images using LablImg [9] python package in YOLO format - one *.txt file extension per image. Each text file consists of one row per object in class, x center, y center, width, height format. The class number starts with zero and the width and height

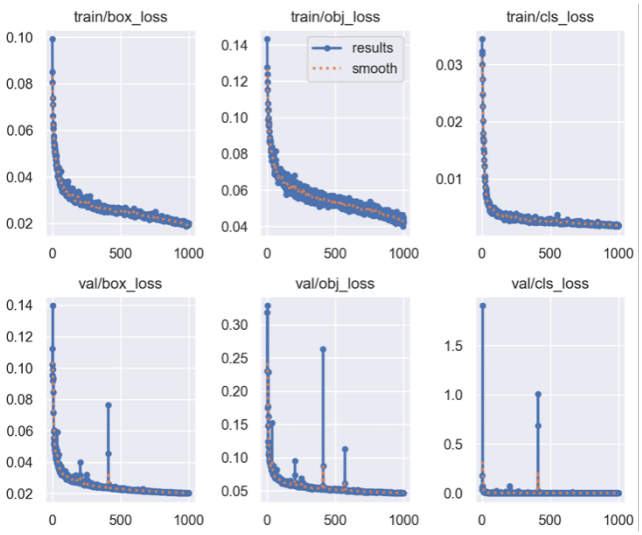

Fig. 5. The Box, object, and class losses vs number of epoch for training and validation process of YOLO-V5 algorithm.

are normalized. After annotating, we divided the dataset into three sets – test, train, and validation with a ratio of 80:10:10. The details of dataset preparation and training process are provided in reference [5]. We trained a YOLO – 5 model with 640 x 640 image resolution. We used the stochastic gradient descent (SGD) as an optimizer, with a learning rate of 0.01, mini- batch size of auto, and 1000 epochs, for the training. The YOLO algorithm uses three loss functions to train the model and calculate the performance – box loss, object loss, and class loss [5], [10]. The box loss, which is computed using IoU (Intersection over Union) loss, measures the difference between predicted bounding box coordinates and ground truth coordinates of the objects. The object Loss, which is computed using Binary Cross-Entropy (BCE) loss, measures the confidence of the model in detecting an object within a bounding box. The class loss, which is calculated using Cross-Entropy (CE) loss, measures the accuracy of the class prediction for each object. The training performance of the model is shown in Fig.5 and table I.

Figure 5 shows that the losses for box, object, and class for both the training and validation dataset decrease with each training cycle (epoch), indicating no over-fitting and the convergence of the model to the global minima. We observe from Table I that the overall precision and recall of the trained model are very high, as are each classes' precision and recall. We therefore conclude that the trained model is effective in detecting almost every instance of a class with a negligible false positive rate. We also observe that the mean average precision (mAP) at an intersection over union (IoU) threshold 50 is close to 1 for all classes and overall; whereas the values of mAP at IoU threshold 50-95 are not as good at mAP50. It implies that more data will e required to achieve a higher value of mAP50-95.

TABLE I
EVALUATION METRICES OF THE YOLO-V5 TRAINING ALGORITHM

Classes	Precision	Recall	mAP50	mAP50-95
All	0.991	0.982	0.991	0.862
Nuts	0.996	0.99	0.993	0.832
Washers	0.996	0.99	0.993	0.832
Screws	0.996	0.99	0.993	0.832

VI. RESULT AND DISCUSSION

After training the model with high precision and recall, we use the system with the trained model to detect and count the mechanical objects in real-time (inference). The results show that the average confidence levels of the model to detect those mechanical objects in real-time are 93.2, 95.8, and 94.2% for nuts, washers, and screws, respectively. Furthermore, Table II shows the performance of the trained model with respect to processing frame per second (FPS) when inferences are done on CPU vs. GPU. As we can see from Table II, the RADX PXIe-GPU can process approximately 10x frames per second than the CPU in the PXIe embedded controller. This difference in throughput illustrates the computational difference between GPUs and CPUs, which is effected via the GPU's parallel computational capabilities.

TABLE II
INFERENCE RESULTS OF THE YOLO-V5 ALGORITHM ON DIFFERENT GPU

Model	CPU	GPU	FPS(CPU)	FPS(GPU)
YOLO5-Small	PXIe -8880 embedded controller (Intel Xeon-E5) with 16GB memory	T600	7.00	55.65
		T1000	6.81	55.56
		A2000	6.67	55.72

VII. CONCLUSION

In this article, we successfully integrated a deep learning-based object detection algorithm in a PXIe ecosystem with PXIe-8880 embedded controller and an NVIDIA-based RADX PXIe-GPU. This system represents a typical PXIe system found throughout the T&M industry and merely reflects the addition of the PXIe-GPU and DL-based computer vision software to enhance its throughput capabilities. We achieved overall high precision, recall, mPA50, and mPA50- 95 on training a model. We also obtained a high frame rate during inference, which is of over 55 FPS on the GPU. This implementation opens and extends the capability of the PXIe system using PXIe-GPU and PXIe-SSD modules.

ACKNOWLEDGMENT

We would like to thank RADX Technologies for support, guidance and providing us with the Catalyst and Trifecta PXIe-GPU Modules based on NVIDIA T600, T1000, and RTX A2000 GPUs as well as the Trifecta PXIe-SSD-RM Module for this effort.

REFERENCES

[1] https://www.ni.com/en/shop/pxi/introduction-to-the-pxi-architecture.html.
[2] https://www.ni.com/docs/en-US/bundle/pxie-1095-features/page/pxi-express-peripheral-slots.html.
[3] O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., ... & Walsh, J. (2020). Deep learning vs. traditional computer vision. In Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference (CVC), Volume 1 1 (pp. 128-144). Springer International Publishing.
[4] Tang, C., Feng, Y., Yang, X., Zheng, C., & Zhou, Y. (2017, July). The object detection based on deep learning. In 2017 4th international conference on information science and control engineering (ICISCE) (pp. 723-728). IEEE.
[5] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788)..
[6] https://www.radxtech.com/wp-content/uploads/2023/04/RADX-Product-Overview-19MAR2024-V2.10-for-Website.pdf
[7] https://www.nvidia.com/en-us/technologies/
[8] Kirk, D. B., & Wen-Mei, W. H. (2016). Programming massively parallel processors: a hands-on approach. Morgan kaufmann.
[9] Tzutalin. LabelImg. Git code (2015). https://github.com/tzutalin/labelImg.
[10] Xu, Q., Zhu, Z., Ge, H., Zhang, Z., & Zang, X. (2021). Effective face detector based on yolov5 and superresolution reconstruction. Computational and mathematical methods in medicine, 2021, 1-9.