Workshop Proceedings of the 19th International Conference on Intelligent Environments (IE2023)
G. Bekaroo et al. (Eds.)

doi:10.3233/AISE230017

Real-World Applications of Machine Learning

Syed Tahir Hussain RIZVI

Abstract. From the real-time forecasting of events to Visual analysis tasks, the state-of-the-art machine learning algorithms exhibit unmatched performance. Furthermore, with the ongoing traction of embedded computing, the deployment of machine learning algorithms on mobile devices is receiving increasing attention. There are numerous practical applications where hand-held devices having machine learning methods can be more useful due to their compact size and integrated resources. However, for the realization of ML methods on embedded devices, either the used algorithm should be less computationally complex or there should be some efficient way to implement a state-of-the-art algorithm on a less-powerful embedded device. In this paper, different approaches for reducing the computational complexity of a machine learning-based computer vision application are presented that can be helpful to make other such algorithms applicable on the embedded devices. Results show that the hardware architecture based exploitation can further improve the performance of an existing framework.

Keywords. Artificial intelligence, machine learning, deep learning, embedded devices, optimization, predictors

1. Introduction

Machine Learning is an ever-growing field with real-time applications reaching out into daily life [2]. Machine Learning has already enjoyed great success in various applications, and its applications are continuously increasing due to the development of new algorithms and methods [7,8,6].

In this work, efficient realization of visual analysis is presented that can be helpful for exploration of other efficient machine learning methods or efficient approaches to implement the required algorithm on an embedded system.

Particularly, the real-world application of machine learning that would be discussed in next section is realization of "Computer Vision Algorithms on Embedded Systems". A highly precise and computationally demanding CV system can be deployed on an embedded system by simplifying the flow of deep architectures developed for powerful desktop and server environments.

2. Efficient Realization of ML Models

This section presents a brief overview of different approaches that can reduce the computational complexity of an existing ML framework and can make their inference realizable on low-powered devices. Furthermore, it is proposed to exploit the hardware architecture based computational resources that can significantly improve the performance of existing implementations.

2.1. CV Algorithms on Embedded System

Utilizing a framework based on CUDA (Compute Unified Device Architecture) or OpenCL (Open Computing Language), it is possible to implement deep learning models on low-powered embedded platforms by taking advantage of pre-trained networks [1,4,5].

Training of computationally and memory-intensive deep classifiers can be accomplished with ease by utilizing powerful GPU servers or desktop workstations, frequently in multi-GPU setups. After training a network, its learned parameters can be imported into an optimized framework specifically designed to replicate the trained neural architectures on embedded platforms [11,3]. This enables the deployment of the trained models for real-world classification tasks. One such implementation is selected in this paper to discuss a few approaches that can be helpful to reduce its computational complexity [10]. The convolution layer is a crucial and computationally demanding component of neural classifiers, responsible for extracting features from input data. Various methods can be employed to compute convolutions, such as the Winograd minimal filtering algorithm, lookup table approach, and fast Fourier transformation-based techniques. The selected CUDA-based framework utilizes a convolution approach that relies on matrix multiplication where the input image and filters are arranged as matrices. These matrices are then transferred to the GPU unified memory to eliminate overhead caused by extra memory transfers and the multiplication of these matrices is performed at a significantly faster speed [9].

Memory allocation schemes such as unified or pinned data allocation can be utilized to optimize the flow of specific layers or the entire framework. Moreover, the developed frameworks or implemented functions can leverage hardware-specific resources to maximize their performance. One such example is the utilization of the Compute capability (C.C), which determines the available features and limitations of the GPU hardware, allowing for efficient acceleration.

Figure 1 shows an efficient approach used by the mentioned framework to use the best available resources depending on the compute capability of the used GPU device. For GPUs with advanced compute capabilities, a high-performance library such as cuBLAS (CUDA Basic Linear Algebra Subroutines) is utilized for matrix multiplication. On the other hand, devices with lower compute capabilities employ shared memory-based matrix multiplication (SAMM) to accelerate the convolutional layer.

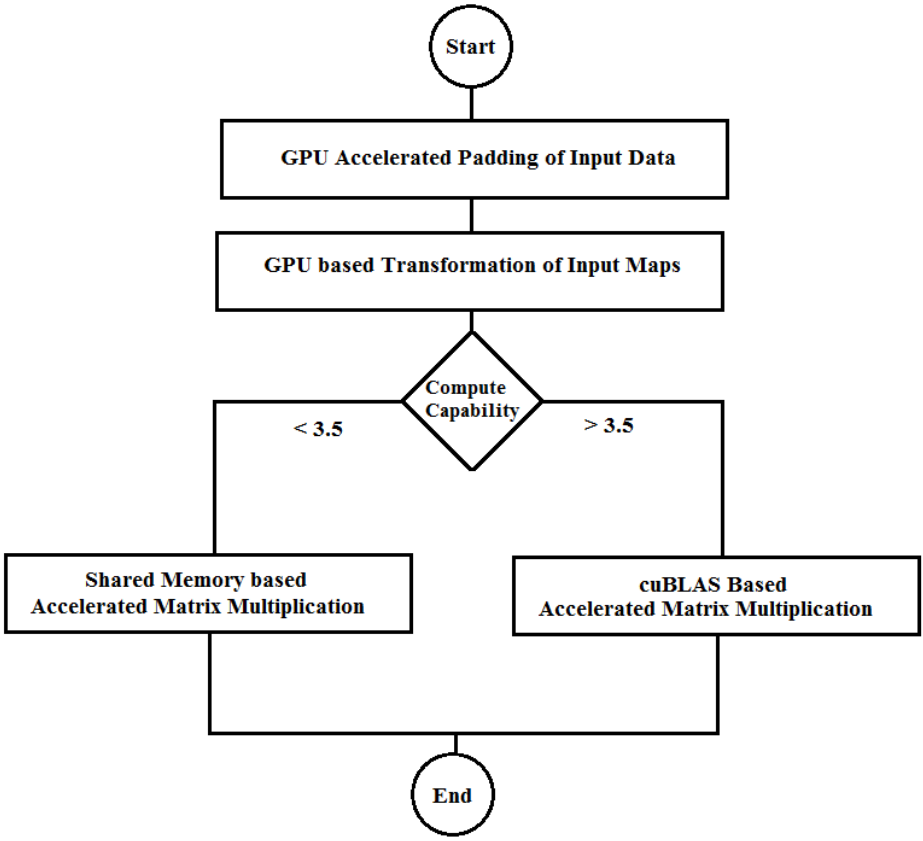

Figure 1. Flow of Hardware-dependent matrix multiplication based Convolutional layer with unified memory [10].

2.2. Hardware Architecture based Exploitation

Furthermore, architecture based exploitation of resources can also significantly improve the performance of such framework.Newer GPU architectures offer support for half-precision floating-point (FP16) data storage and arithmetic operations, which can significantly enhance the performance of training and inference in deep classifiers. The primary benefit of utilizing the half-precision data format, compared to the 32-bit single-precision format, is that it reduces storage and bandwidth requirements by half, while maintaining accuracy with little to no noticeable loss. By converting the framework and variables, including input, output, and trainable parameters, to a half-precision format, the arithmetic complexity and storage requirements can be minimized significantly for embedded platforms equipped with modern GPUs like Pascal that support FP16 calculations. Figure 2 provides a visual representation of the composition of various floating point representations.

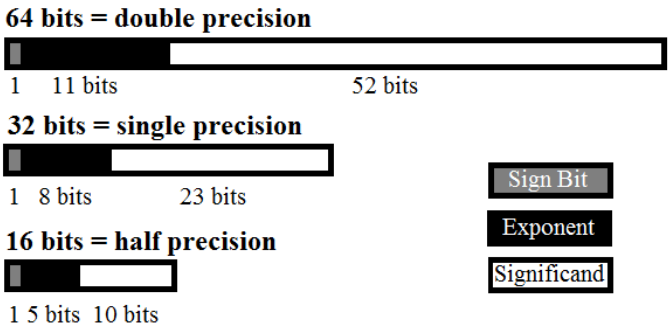

Figure 2. Various Floating Point Representations.

3. Results

The performance of exisitng CUDA-based framework and its results discussed in [10] are further improved by using discussed hardware architecture based exploitation approach. The hardware platform employed for experiments is the Nvidia Jetson TX1 embedded board, which features a quad-core ARM Cortex A57 CPU, an Nvidia Maxwell GPU with 256 CUDA cores, and 4 GB of shared RAM. It is also equipped with 16 GB of embedded MultiMediaCard (eMMC) storage. Since the Jetson TX1 board is equipped with a Pascal GPU and supports half-precision storage and arithmetic operations, it is possible to achieve additional performance speedup by converting the proposed framework to FP16 format. This involves converting all imported parameters and allocations from float- to half-precision data types, allowing the utilization of the half-precision General Matrix-Matrix Multiply (HGEMM).

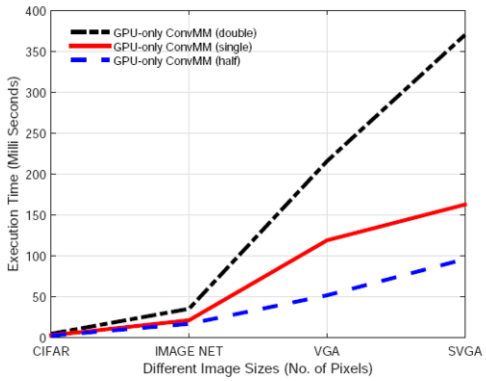

Image Size	GPU-only ConvMM Layer		
	Double	Single	Half
	(Milliseconds)		
CIFAR (32 × 32 × 16)	5.12	3.20	**2.76**
ImageNet (224 × 224× 16)	35.61	21.63	**17.20**
VGA (640 × 480 × 16)	215.74	119.17	**52.03**
SVGA (800 × 600 × 16)	369.91	162.80	**96.26**

Figure 3. Comparison of double-, single- and half-precision ConvMM layers on Jetson TX1 Board, best results are written in bold.

It can be visualized from Figure 3 that the GPU-only ConvMM layer in half-precision demonstrates a speed increase of approximately 4 times compared to the double-precision version. Moreover, it exhibits a speed improvement of 2 times when compared to the single-precision scheme.

Figure 4 further validates that the half-precision deep classifiers are significantly faster than the other versions and can be used to accelerate the performance of a framework. A further analysis of employing half-precision cuBLAS-Accelerated Matrix Multiplication Convolution (ConvCAMM) and unified memory in existing framework also verifies the effectiveness of applied approaches.

Model	Layers	GPU-only ConvMM Layer			Unified ConvCAMM
		Double	Single	Half	Half
		(Milliseconds)			
Alex's CIFAR-10	5	107.03	76.36	**50.41**	**12.85**
OverFeat	8	1924.41	1212.74	**836.03**	**88.02**
ResNet-34	34	1217.33	887.53	**724.47**	**362.76**

Figure 4. Classification time of double-, single- and half-precision deep models on Jetson TX1 Board along with half-precision Unified ConvCAMM layer, best results are written in bold.

4. Conclusion

This paper introduces a proposal that highlights the potential enhancements in performance for an existing framework through various approaches. These approaches include an optimized data transfer scheme, hardware-dependent matrix multiplication, and the exploitation of GPU architecture-based resources. The results obtained validate the effectiveness of the proposed hardware architecture-based exploitation scheme, demonstrating its ability to achieve real-time image classification on embedded platforms.

By reducing the computational complexity of the convolution operation, the performance of the current framework can be further enhanced. One way to achieve this is by implementing Winograd's minimal filtering technique, which effectively minimizes the arithmetic complexity of the convolution operation when applied to smaller tiles. Various approaches can be adopted to incorporate this technique and further optimize the framework's performance.

References

[1] Mhd Rashed Al Koutayni, Gerd Reis, and Didier Stricker. Deepedgesoc: End-to-end deep learning framework for edge iot devices. *Internet of Things*, 21:100665, 2023.

[2] Antonio Coronato and Muddasar Naeem. A reinforcement learning based intelligent system for the healthcare treatment assistance of patients with disabilities. In *International Symposium on Pervasive Systems, Algorithms and Networks*, pages 15–28. Springer, 2019.

[3] Ashkan B Jeddi, Abdollah Shafieezadeh, and Roshanak Nateghi. Pdp-cnn: A deep learning model for post-hurricane reconnaissance of electricity infrastructure on resource-constrained embedded systems at the edge. *IEEE Transactions on Instrumentation and Measurement*, 2023.

[4] Junwen Liu, Ziyun Xiao, Shiyong Lu, Dunren Che, Ming Dong, and Changxin Bai. Infrastructure-level support for gpu-enabled deep learning in dataview. *Future Generation Computer Systems*, 141:723–737, 2023.

[5] Yukui Luo, Shuai Li, Kuangyuan Sun, Raul Renteria, and Ken Choi. Implementation of deep learning neural network for real-time object recognition in opencl framework. In *2017 International SoC Design Conference (ISOCC)*, pages 298–299. IEEE, 2017.

[6] M. Naeem, S. T. H. Rizvi, and A. Coronato. A gentle introduction to reinforcement learning and its application in different fields. *IEEE Access*, 8:209320–209344, 2020.

[7] Muddasar Naeem and Antonio Coronato. An ai-empowered home-infrastructure to minimize medication errors. *Journal of Sensor and Actuator Networks*, 11(1):13, 2022.

[8] Muddasar Naeem, Antonio Coronato, and Giovanni Paragliola. Adaptive treatment assisting system for patients using machine learning. In *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 460–465. IEEE, 2019.

[9] Syed Tahir Hussain Rizvi, Gianpiero Cabodi, and Gianluca Francini. Gpu-only unified convmm layer for neural classifiers. In *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 0539–0543, 2017.

[10] Syed Tahir Hussain Rizvi, Gianpiero Cabodi, and Gianluca Francini. Optimized deep neural networks for real-time object classification on embedded gpus. *Applied Sciences*, 7(8):826, 2017.

[11] Abdul Sami, Ali Asif, Muhammad Imran, Farah Aziz, and Muhammad Yasir Noor. Artificial neural network and dataset optimization for implementation of linear system models in resource-constrained embedded systems. *Expert Systems*, page e13142, 2023.