ISSN: 2278-4632
Vol-13, Issue-07, July 2023

A COMPARATIVE STUDY OF INFERENCE ENGINE DEVELOPMENT: ACL VS. TENSORFLOW

Bussa Uday Kiran, Research Scholor, Bir Tikendrajit University
Dr. Manukonda Divya Research Supervisor, Bir Tikendrajit University

ABSTRACT

Embedded systems that include deep learning inference engines mark a major step forward for AI research and development. In order to run deep learning models successfully on embedded devices, which have limited CPU resources and strict power limitations, highly optimized inference engines are necessary. Our experience in developing an ACL-based deep learning inference engine from ground up is detailed in this work. Based on the findings, it is more efficient to construct an inference engine from scratch for basic models than to transfer existing frameworks, which goes against the prevailing knowledge. Furthermore, we were able to construct an inference engine that surpasses TensorFlow by 25% with the use of ACL.

Keywords: Inference engine, Embedded,SqueezeNet, TensorFlow, Convolution

I. INTRODUCTION

The exponential growth of AI and ML in the last few years has revolutionized several sectors of the economy via pushing innovation in almost every area of technology. A key component of artificial intelligence (AI) solutions for embedded systems, deep learning inference engines have evolved into a game-changer in this area. Embedded systems are essential to many different kinds of applications, including consumer electronics, automotive systems, industrial automation, and Internet of Things (IoT) devices, despite their restricted computing capabilities and power restrictions. As a result, there are tremendous possibilities and enormous problems at the intersection of deep learning and embedded systems. Within the framework of deep learning, a component known as an inference engine is responsible for applying previously learned models to fresh data in order to provide predictions. The inference step, in contrast to the computationally heavy training phase, which is often carried out on robust server-grade hardware, requires fast execution on the target device. When it comes to embedded systems, this is of the utmost importance since optimizing resources is of the utmost importance. Efficient, low-latency, and power-efficient inference engines are becoming more important as AI moves closer to the edge.

When it comes to embedded systems, inference engines have their own set of challenges and needs. Among them, you might find demanding goals for energy usage, memory limits, and processor power. And because of the importance of latency to both performance and user experience, they often function in real-time settings. There must be a delicate balancing act between computing efficiency and the capacity to provide accurate and dependable forecasts in the design and development of these engines. To adjust these models to suit inside the limited environment is one of the main problems of implementing deep learning models on embedded devices. Modern deep learning models, including RNNs, transformers, and convolutional neural networks (CNNs), often need a lot of memory and processing power to function well. It is very uncommon for these models to have subpar performance or be completely unfeasible owing to resource fatigue when transferred directly to embedded devices. Therefore, in order to prepare these models for embedded deployment, methods such model compression, quantization, pruning, and efficient architecture design are used.

Reducing the size and computing needs of deep learning models is made possible by model compression approaches like as pruning and quantization. To simplify the model without significantly compromising its accuracy, pruning removes unnecessary or insignificant weights from the neural network. In contrast, quantization drastically lowers the memory footprint and computing demand by reducing the accuracy of the model's parameters, usually from 32-bit floating-point to 8-bit integers. Specialized algorithms and efficient neural network designs, including MobileNets and SqueezeNet, have been developed to supplement these approaches; they provide competitive

ISSN: 2278-4632
Vol-13, Issue-07, July 2023

performance with much reduced resource needs. The incorporation of software and hardware co-design into embedded inference engines is another crucial component. It is crucial to optimize software algorithms and hardware design simultaneously in embedded systems due to their heterogeneous nature, which may include CPUs, GPUs, DSPs (Digital Signal Processors), and specialized AI accelerators. To achieve peak performance while using as little power as possible, the inference engine is co-designed with each hardware component to play to their own strengths. Popular frameworks like TensorRT and TVM make it easy to optimize for certain hardware capabilities by letting developers customize the inference process.

Additionally, low-latency and robust inference is required since many embedded applications operate in real-time. For example, autonomous cars must interpret data from images and sensors in real-time in order to make split-second judgments, the failure to which might have disastrous repercussions. In the medical field, wearable gadgets are also required to analyze biosignals in real-time so that prompt notifications and actions may be given. Therefore, it is crucial for embedded inference engines to have excellent responsiveness and efficiency. The dependability and security of embedded systems are of the utmost importance. It is crucial to guarantee the security and integrity of the inference process since these systems often run in mission-critical applications. To prevent manipulation and illegal access, methods including encrypted model storage, secure boot, and runtime integrity checks are used. Moreover, fault tolerance methods are included to provide dependable functioning regardless of software or hardware issues. "Recent developments in edge computing have also helped with the implementation of inference engines for deep learning on embedded devices." Reduced latency, conserved bandwidth, and improved privacy are all benefits of edge computing, which processes data locally on the device instead of depending on resources in the cloud. Because embedded inference often requires localized, real-time data processing, this decentralization is in line with its aims. New opportunities for cutting-edge applications in several fields arise from the complementary nature of edge computing and embedded inference.

II. REVIEW OF LITERATURE

Stefani, Domenico et al., (2022) Recent progress in deep learning has shown significant promise for audio applications, enhancing the precision of prior methods for tasks such as music transcription, beat recognition, and real-time audio processing. Furthermore, the presence of more advanced embedded processors has prompted deep learning framework creators to build software specifically designed to efficiently execute pre-trained models in situations with limited resources. "Consequently, the use of deep learning on embedded devices and audio plugins has seen a broader adoption." Nevertheless, there is increasing uncertainty around deep learning inference engines, namely about their real-time capabilities and resource efficiency. This work provides a comprehensive analysis of four deep learning inference engines, namely TensorFlow Lite, TorchScript, ONNX Runtime, and RTNeural, in terms of their suitability for real-time audio categorization on the CPU of an embedded single-board computer. The results indicate that all inference engines are capable of executing neural network models in real-time when acceptable coding techniques are used. However, the execution time may vary depending on the specific engine and model being used. Significantly, our findings indicate that the majority of the less-specialized engines have excellent adaptability and may be successfully used for real-time audio classification, yielding somewhat superior outcomes compared to a real-time-specific strategy. Conversely, more specialized solutions might provide a streamlined and simple option when less adaptability is required.

Seng, Kah & Ang, Li-minn. (2022)In recent years, there has been a rise in the use of advanced artificial intelligence (AI) and machine learning methods. These techniques are being used on cloud servers and incorporated into edge computing devices to enable Internet of Things (IoT) and mobile apps. It is crucial to acknowledge that the deployments of embedded intelligence (EI) on edge devices and cloud servers vary significantly in terms of aims, models, platforms, and research issues. This study provides a thorough examination of Emotional Intelligence (EI) from four different perspectives: (1) Initially, the paper introduces and examines the most advanced method for

ISSN: 2278-4632
Vol-13, Issue-07, July 2023

evaluating EI using a specific set of criteria; (2) Next, the paper explores EI in relation to both cloud server accelerators and low-complexity edge devices; (3) Furthermore, the paper classifies and discusses the different techniques for EI based on system, algorithm, architecture, and technology levels; and (4) Finally, the paper concludes by summarizing the insights gained and discussing the future prospects of EI, particularly its significant role in emerging technologies and applications like Industry 4.0. This article seeks to provide valuable insights and future possibilities for advancements in this field of research, while also emphasizing the problems faced in implementing these discoveries in real-world scenarios.

Wang, Siqi et al., (2019) IoT Edge intelligence necessitates the execution of Convolutional Neural Network (CNN) inference directly on the edge devices. ARM is large.The LITTLE architecture is crucial to the widespread use of commercial edge devices. The system consists of individual instruction set architecture (ISA) cores that are different from one another, organized into several clusters that are all the same. This allows for balancing power consumption and performance. To achieve maximum throughput, it is anticipated that all cores would be used concurrently for inference. Nevertheless, the significant communication cost associated with parallelizing calculations using convolution kernels across clusters has a negative impact on performance. We provide a different approach dubbed Pipe-it, which utilizes a pipelined architecture to distribute convolutional layers across clusters. This method restricts the parallelization of their respective kernels to the designated cluster. We create a model that predicts the execution time of each layer separately on all allowed core configurations (type and count) using just the descriptors of the convolutional layer. Pipe-it utilizes the predictions to construct a well-balanced pipeline via the use of an efficient algorithm for exploring the design space. Pipe-it typically yields a 39% greater throughput compared to the maximum previous throughput.

Han, Song et al., (2016) Modern deep neural networks (DNNs) contain a large number of connections, reaching hundreds of millions, and need significant processing power and memory. This is a challenge when trying to implement them on embedded devices that have limited hardware resources and power constraints. Although bespoke hardware aids in the calculation process, retrieving weights from DRAM is much more costly than ALU operations, with a difference of two orders of magnitude, and it is the main factor that consumes power. The previously suggested 'Deep Compression' technique enables the complete fitting of massive Deep Neural Networks (DNNs) such as AlexNet and VGGNet into on-chip SRAM. This compression is accomplished by removing unnecessary connections and allowing numerous connections to have the same weight. We present a novel energy-efficient inference engine (EIE) that conducts inference on a compressed network model and enhances the speed of the resultant sparse matrix-vector multiplication via weight sharing. Transitioning from DRAM to SRAM results in a significant energy reduction of 120 times for EIE. Utilizing sparsity leads to a 10-fold energy saving. Implementing weight sharing contributes to an 8-fold energy reduction. Additionally, skipping zero activations from ReLU further saves 3 times the energy. When tested on nine deep neural network benchmarks, the EIE system demonstrated a speed improvement of 189 times compared to the CPU implementation and 13 times compared to the GPU version of the same deep neural network without compression. The EIE has a processing power of 102 giga operations per second (GOPS) while operating on a compressed network, which is equivalent to 3 tera operations per second (TOPS) on an uncompressed network. It can process fully connected (FC) layers of the AlexNet model at a rate of 1.88×10^4 frames per second, while using just 600 milliwatts (mW) of power. It is 24,000 times more energy efficient than a CPU and 3,400 times more energy efficient than a GPU. When comparing DaDianNao to EIE, it is found that EIE has much greater throughput, energy efficiency, and area efficiency. Specifically, EIE has 2.9 times better throughput, 19 times better energy efficiency, and 3 times better space efficiency.

Ramadoss, Ashok Kumar & Krishnaswamy, Marimuthu (2014)The objective of this study is to identify a suitable fuzzy implication for approximation reasoning in different scenarios and to address reasoning problems using fuzzy production rules, also known as approximate reasoning, in the development of inference engines for fuzzy expert systems. When the knowledge domain is removed from an expert system, what remains is an expert system shell. The expert system shell may

ISSN: 2278-4632
Vol-13, Issue-07, July 2023

be used to several knowledge domains, not limited to just one. A reusable inference engine, integrated inside a suitable expert system shell, may be used across many areas of knowledge for different expert systems, in collaboration with relevant human experts. This study aimed to identify significant criteria for evaluating and comparing various fuzzy implications. This study employs an explanatory interface to enhance communication between the user and the expert system. Modus ponens is used to evaluate the appropriate production norms. This study has shown that experienced individuals are capable of performing and acknowledging that the domains of fuzzy systems and neural networks are closely linked. In this study, it has been shown that the process of fuzzification is very beneficial in the field of humanoid robotics. Fuzzy sets are used to create membership functions for relevant fuzzy sets and other context-dependent entities, based on sample data. The impetus for approximating fuzzy systems using neural networks is derived from the neural networks' intrinsic capacity to do extensive parallel processing of input. This is pertinent to fuzzy controllers and particularly applicable to fuzzy expert systems that handle substantial quantities of fuzzy inference rules in this real-time investigation.

Iglesias, Josue et al., (2012) Embedding context management in devices with limited resources, such as mobile phones, autonomous sensors, or smart objects, requires data modeling and reasoning to be lightweight. This paper examines the current advancements in data representation and reasoning tools for embedded mobile reasoning. It introduces a light inference system (LIS) that aims to streamline embedded inference processes by providing a range of functionalities to prevent redundancy in context management operations. The system is a component of a mobile software framework that is designed to simplify the development of context-aware apps. It separates the gathering of sensor data and context processing from the application logic. LIS is a software system that consists of many modules. These modules provide lightweight tools for managing ontology data and performing rule-based reasoning. Additionally, LIS is designed to be compatible with Java-enabled handheld devices. Data management and reasoning procedures are specifically built to handle a comprehensive ontology that facilitates communication between different components of the framework. Both the applications using the framework and the framework components themselves have the ability to configure the rule and query sets in order to obtain the necessary information from LIS. To evaluate the characteristics of LIS in an actual application situation, a 'Activity Monitor' has been created and executed. This is a personal health-persuasive application that offers feedback on the user's lifestyle by integrating information from both physical and virtual sensors. In this particular use scenario, LIS is used to promptly assess the user's level of engagement, establish if it is appropriate to initiate notifications, and choose the most suitable interface or channel for delivering these context-aware warnings.

III. RESEARCH METHODOLOGY

We have constructed a SqueezeNet mechanism by using the fundamental components of ACL. Now, we may explore and compare the efficiency of TensorFlow with that of ACL. In order to guarantee a fair comparison, we activated the ARM NEON vector computation optimization feature in TensorFlow. Additionally, while constructing our SqueezeNet engine, we deliberately opted for NEON-enabled building blocks. By ensuring that both engines employ the NEON vector computation, we guaranteed that any disparity in performance would be solely attributable to the platform itself.

IV. RESULTS AND DISCUSSION

Table 1 shows that we used four cores on our Zuluko platform to run SqueezeNet with TensorFlow. An RGB picture with dimensions 227 by 227 took around 420 milliseconds to compile. A 25% speedup was achieved by processing the same picture in only 320 ms using the SqueezeNet engine constructed from ACL. "We took it a step further and split the processing time in half, with half going to pooling and soft-max and the other half to convolution, RELU, and concatenate; this helped us identify where the performance boost was coming from." Compared to TensorFlow, our SqueezeNet engine performs 23% better in Group 1 and 110% better in Group 2, according to the

ISSN: 2278-4632
Vol-13, Issue-07, July 2023

breakdown data. When it comes to resource use, operating on TensorFlow typically uses 75% of the CPU and roughly 9 MB of RAM. Simultaneously, when operating on ACL, the CPU utilization averages 90% and memory utilization is about 10%.

The speed boost might be due to two factors. Firstly, TensorFlow depended on the ARM compiler to achieve NEON optimization, while ACL's operators were built utilizing NEON intrinsic operators directly, therefore ACL offers superior NEON optimization. Second, there is probably some performance overhead introduced by the TensorFlow platform itself.

Table 1: TensorFlow vs. ACL

Particulars	TensorFlow	ACL
Convolution, RELU and Concatenate	280	220
Pooling and soft-max	350	280

The next step was to see whether TensorFlow could achieve better performance than the inference engine that was created on top of ACL. We made an effort in this direction by using the optimization method of vector quantization. To optimize for both speed and accuracy, this optimization mostly use 8-bit weights. Furthermore, vector calculations allow us to process many data units with a single instruction when 8-bit weights are used. Nevertheless, re-quantize and de-quantize procedures are introduced as a result of this optimization, which is a cost. Table 2 shows the results of comparing the two scenarios with and without quantization. We were able to get a 25% improvement in convolution performance by using vector quantization. Yet, due to the de-quantization and re-quantization processes, it adds substantial overhead. In fact, it adds over 100 milliseconds to the whole inference time.

Table 2: Quantization performance

Particulars	quantization	without quantization
convolution	210	280
others	500	450

V. CONCLUSION

Improving AI's performance in low-resource settings requires the creation and implementation of embedded systems-specific deep learning inference engines. This work has covered all the bases when it comes to developing embedded device inference engines that are efficient, dependable, and latency-sensitive. Model compression, quantization, pruning, and the building of customized neural network designs like MobileNets and SqueezeNet are novel solutions that are necessary for the particular problems presented by restricted processing capacity, memory limits, and energy efficiency. In order to optimize inference engines to fully use the capabilities of heterogeneous embedded platforms, such as CPUs, GPUs, DSPs, and AI accelerators, software and hardware co-design has become an essential method. Developers have access to tools like TensorRT and TVM, which allow them to tailor the inference process to the unique hardware configurations of embedded devices.

REFERENCES: -

1. Zhang, Zhaoyun & Li, Jingpeng. (2023). A Review of Artificial Intelligence in Embedded Systems. Micromachines. 14. 897. 10.3390/mi14050897.
2. Stefani, Domenico & Peroni, Simone & Turchet, Luca. (2022). A Comparison of Deep Learning Inference Engines for Embedded Real-time Audio Classification.Proceedings of the 25th International Conference on Digital Audio Effects (DAFx20in22), Vienna, Austria.
3. Seng, Kah & Ang, Li-minn. (2022). Embedded Intelligence: State-of-the-Art and Research Challenges. IEEE Access. 10. 1-1. 10.1109/ACCESS.2022.3175574.
4. Shumba, Angela-Tafadzwa & Montanaro, Teodoro & Sergi, Ilaria & Fachechi, Luca & Vittorio, Massimo & Patrono, Luigi. (2022). Embedded Machine Learning: Towards a Low-Cost Intelligent IoT edge. 1-6. 10.23919/SpliTech55088.2022.9854248.
5. Lim, Seung-Ho & Kang, Shin-Hyeok & Ko, Byeong-Hyun & Roh, Jaewon & Lim, Chaemin

ISSN: 2278-4632
Vol-13, Issue-07, July 2023

& Cho, Sang-Young. (2022). An Integrated Analysis Framework of Convolutional Neural Network for Embedded Edge Devices. Electronics. 11. 1041. 10.3390/electronics11071041.

6. Nordby, Jon. (2021). emlearn: Machine Learning inference engine for Microcontrollers and Embedded Devices. 10.5281/zenodo.2589394.
7. Chen, Tse-Wei & Tao, Wei & Wang, Deyu & Wen, Dongchao & Osa, Kinya & Kato, Masami. (2021). Hardware Architecture of Embedded Inference Accelerator and Analysis of Algorithms for Depthwise and Large-Kernel Convolutions.
8. Vreča, Jure & Sturm, Karl & Gungl, Ernest & Merchant, Farhad & Bientinesi, Paolo & Leupers, Rainer & Brezočnik, Zmago. (2020). Accelerating Deep Learning Inference in Constrained Embedded Devices Using Hardware Loops and a Dot Product Unit. IEEE Access. 8. 165913-165926. 10.1109/ACCESS.2020.3022824.
9. Wang, Siqi & Ananthanarayanan, Gayathri & Zeng, Yifan & Goel, Neeraj & Pathania, Anuj & Mitra, Tulika. (2019). High-Throughput CNN Inference on Embedded ARM big.LITTLE Multi-Core Processors. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. PP. 1-1. 10.1109/TCAD.2019.2944584.
10. Yoo, Seung-mok & Cho, Changsik & Lee, Kyung & Park, Jaebok & Jin, Seok & Lee, Youngwoon & Kim, Byung-Gyu. (2019). Structure of Deep Learning Inference Engines for Embedded Systems. 920-922. 10.1109/ICTC46691.2019.8939843.
11. Dey, Swarnava & Mukherjee, Arijit & Pal, Arpan & Purushothaman, Balamuralidhar. (2019). Embedded Deep Inference in Practice: Case for Model Partitioning. 25-30. 10.1145/3362743.3362964.
12. Liang, Yi & Cai, Zhipeng & Yu, Jiguo & Han, Qilong & Li, Demin. (2018). Deep Learning Based Inference of Private Information Using Embedded Sensors in Smart Devices. IEEE Network. 32. 8-14. 10.1109/MNET.2018.1700349.
13. Han, Song & Liu, Xingyu & Mao, Huizi & Pu, Jing & Pedram, Ardavan & Horowitz, Mark & Dally, William. (2016). EIE: Efficient Inference Engine on Compressed Deep Neural Network. ACM SIGARCH Computer Architecture News. 44. 10.1145/3007787.3001163.
14. Lane, Nicholas & Bhattacharya, Sourav & Georgiev, Petko & Forlivesi, Claudio & Kawsar, Fahim. (2016). Demo: Accelerated Deep Learning Inference for Embedded and Wearable Devices using DeepX. 109-109. 10.1145/2938559.2949718.
15. Ramadoss, Ashok Kumar & Krishnaswamy, Marimuthu. (2014). Embedding inference engine in fuzzy expert robotic system shell in a humanoid robot platform for selecting stochastic appropriate fuzzy implications for approximate reasoning. Artificial Life and Robotics. 20. 13-18. 10.1007/s10015-014-0189-2.
16. Iglesias, Josue & Barbolla, Ana & Tarrío, Paula & Casar, José & Martín, Henar. (2012). Design and validation of a light inference system to support embedded context reasoning. Personal and Ubiquitous Computing. 16. 10.1007/s00779-011-0447-4.