Towards a Performance Portable Productive Library for Convolutional Neural Network Deployment

Changxin Li
Computer and Data Sciences
Case Western Reserve University
Cleveland, USA
cxl1492@case.edu

Sanmukh Kuppannagari
Computer and Data Sciences
Case Western Reserve University
Cleveland, USA
sanmukh.kuppannagari@case.edu

***Abstract*—Convolutional Neural Networks (CNN) have demonstrated significant success in a variety of image and video processing applications. Extensive research has been conducted in the development of convolution algorithms for various use cases to accelerate CNNs. However, with the increasing heterogeneity in hardware platforms and the plethora of available convolution algorithms, it is non-trivial to achieve performance portability for the deployment of CNN models.**

In this work, we describe one portion of our project whose overarching goal is to develop a performance portable productive library for CNN deployment, targeting a variety of CNN models over a variety of heterogeneous platforms. Specifically, we discuss the development of our SYCL based convolution algorithm library that achieves high performance on both GPU and FPGA platforms for convolution algorithms. We conduct extensive experiments to compare the execution times of various convolution algorithms on the two platforms using pytorch and Intel extensions for pytorch as the baselines and demonstrate significant speedups in convolution layer execution.

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Convolutional Neural Networks (CNN) are deep learning techniques that have demonstrated tremendous effectiveness in a variety of application domains that rely on image or video processing. High impact applications such as facial recognition for security and access control [1], medical image processing [2], automated quality control of wafer design [3], material characterization [4], etc. are just a few examples of the successes of CNNs.

In recent years, Vision Transformer based foundation models have demonstrated superior performance than CNNs [5], thereby, challenging their dominance. However, despite that, CNNs have retained their popularity as unlike general purpose extremely large scale one-size-fits-all Vision Transformer models, a variety of application specific CNNs models have been developed over the past decade. For example, models such as variants of mobilenets are extremely suitable for edge devices (due to depthwise convolution) [6], while models such as U-nets are highly popular in medical image segmentation applications [7]. Furthermore, researchers have started developing CNN models that have the ability to seamlessly scale as Vision Transformer models using deformable convolutions [8].

The versatility of CNNs can be attributed to the extensive research that has been conducted on developing Convolution algorithms — the most computationally intensive kernel of CNNs, targeting a variety of use cases [9]. For example, IM2COL, was developed to represent convolution layers as matrix multiplications to enable utilization of BLAS library and significantly enhance the portability of these models [10]. KN2ROW was developed to reduce the memory requirements of IM2COL while depthwise convolution was developed to reduce both memory and computational requirements making it suitable for edge devices [11]. Deformable convolutions [12] have been developed recently to mitigate the limitations of fixed geometric structures and are also being used to develop large-scale CNN networks that are similar in capabilities of emerging Vision Transformer models [8].

Simultaneously, another visible trend is the proliferation of AI accelerators as increasingly customized accelerators are being developed for various use cases [13]. From high performance data centers to low powered edge devices, GPUs and FPGAs spanning across this entire spectrum are available in the market [13]. Processing In Memory (PIM) based architectures are targeting low latency, low power use cases [14]. Additionally, startups that are developing custom ASIC based AI accelerators such as Cerebras (for high performance datacenters) [15] have proliferated in recent years.

This immense heterogeneity poses significant challenges with respect to the 3Ps — Performance, Portability, and Productivity, both for the end users — who apply CNN models to application domains as well as systems developers/researchers — who are responsible for developing frameworks that enable high performant deployment. Current AI frameworks such as pytorch default to using a single variant of convolution, which while *Portable* does not necessarily provide the best Performance. To enable *Performance*, end users are required to understand the tradeoffs of different convolution algorithms, leading to a drop in *Productivity*. From a systems developer/researcher point of view, while customized CNN implementations exist for various use cases [11], [16], [17] and some frameworks exist that enable dynamic selection of convolution algorithms for improved performance for a small set of convolution algorithms [9], developing a 3P CNN library is still a surmounting task due to the following reasons: while the research in developing novel convolution algorithms is extensive, transforming the research artifacts

into fully functional, thoroughly tested code that works on a variety of hardware platforms limits portability and productivity. Moreover, optimal selection of layer specific convolution algorithms to obtain high performance will require testing a combinatorially large space of design points (combinatorial in hardware parameters, layer parameters, and number of convolution algorithms), leading to limited performance and/or productivity.

The overarching goal of this project (Figure 1) is to develop a *Performance Portable Productive Library* for Convolution Neural Networks targeting a variety of CNN models over a variety of heterogeneous devices, while easily amenable to inclusion of ever expanding convolution algorithms. Additionally, the library should provide a productive environment for the end application user by requiring little to no change to their pytorch code.

In this paper, we describe a portion of the completed work towards building the library (highlighted in red dashed rectangle in Figure 1). Specifically, the contributions of this work are:

- We build a high performant, portable library of convolution algorithms using SYCL/oneAPI targeting GPUs and FPGAs.
- We develop a pytorch based framework that enables execution of a convolution layer using a selected convolution algorithm.
- We conduct extensive experimentation to compare the performance of CNN deployments using our framework on GPU and FPGA, and using pytorch on GPU. We demonstrate upto 8x speedup using GPUs and 5x speedup using FPGAs when considering the overall *convolution layers* execution time against pytorch on GPU as the baseline.
- We also compare the performance of our framework against Intel Extensions for Pytorch [18], which is an optimized version of pytorch, on Intel GPU and FPGA. The CNNs deployed using our framework demonstrate a speedup of as high as 1.5x against Intel extensions for pytorch when considering the overall *convolution layers* execution time.
- We conduct experiments to compare the *end to end latency* of various CNN models using pytorch and Intel extensions for pytorch on GPU as baselines, versus our framework targeting a GPU and FPGA to inform our immediate next step of optimizing the additional overheads of our framework.

II. Background and Related Works

A. Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a type of Deep Neural Networks that use Convolution Layers as the core technique for feature extraction. Each convolution layer is characterized by the following parameters [19]: Input Feature Dimensions: $H_{in} \times W_{in} \times C_{in}$, where $H_{in} \times W_{in}$ denote the image sizes and C_{in} denotes the channel width - basically, the number of images; Output Feature Dimensions: $H_{out} \times W_{out} \times C_{out}$, where $H_{out} \times W_{out}$ denote the image sizes and C_{out} denotes the channel width; and Kernel Dimensions: $k \times k \times C_{in} \times C_{out}$. The operation performed by convolution layer can then be represented by the following equation:

$$O(i,j,c_{out}) = \sum_{c_{in}=1}^{C_{in}} \sum_{i=1}^{H_{in}} \sum_{j=1}^{W_{in}} \sum_{k1=1}^{k} \sum_{k2=1}^{k} I(i+k1, j+k2) * K(k1, k2, c_{in}, c_{out})$$

Here I, O, K denote the input feature image, output image, and kernel respectively. In practice, appropriate padding and striding are incorporated as suitable to obtain the output image.

B. Research on Convolution Algorithms

Extensive research has been conducted on developing algorithms to optimize the convolution layers for different objectives. IM2COL was one of the earliest works that modeled the convolution operation as matrix multiplications to enable the use of BLAS libraries optimized for devices [10]. KN2ROW was developed as an improvement upon IM2COL to reduce its memory requirements [11]. DEPTHWISE convolution was developed to reduce the computational complexity of the convolution layer and has been especially effective in edge settings [6]. Other algorithms such as direct convolution [20], scalar matrix multiplication [16], indirect convolution [21], deformable convolution [12], etc. have also been developed for various use cases.

C. Related Works on CNN deployment

A variety of works have focused on accelerating the deployment of CNNs on FPGAs [22], GPUs [23], and ASICs [24]. However, most of these work focus on hardware specific optimizations with the main goal of achieving best performance — execution time, energy-efficiency, etc. at the cost of portability.

Work such as FPG-AI [25], BODA-RTC [26], Intel Pytorch Extensions are a few works that have focused on the portability aspect of CNN deployment. In FPG-AI, authors developed a performance portable CNN deployment framework for a variety of FPGA devices. They achieved this by developing a template Convolution IP and using design space exploration to customize the template for the target FPGA. In BODA-ETC [26], authors developed a deployment framework that implements the key operations of CNNs using openCL to enable deployment on a variety of GPU platforms. They only support the IM2COL algorithm implementation of convolution. The limitations with these frameworks are they achieve limited portability on either FPGA or GPU.

Intel Extension for Pytorch [18] is a DNN optimization framework from Intel that currently support deployment of CPU and GPU platforms. We are not aware of this framework supporting FPGAs currently. However, given that the framework leverages SYCL/onAPI [27] as one of the underlying library, future iterations may support deployment on FPGA platforms.

In summary, to the best of our knowledge, there does not exist a performance portable framework for deploying CNNs on a variety of target platforms, thereby, motivating the need for this project.

D. SYCL/oneAPI

SYCL programming language was developed by Khronos group to address the immense heterogeneity that is being exhibited by modern computing platforms due to the integration of accelerators such as FPGA, GPU, and custom AI processors with CPU platforms [27]. Intel offers an implementation of SYCL using its oneAPI frameworks and libraries. Using SYCL/oneAPI, a programmer can write the code once and deploy it on a variety of CPU+X platforms, where X can be any accelerator. While the program can run with a reasonable performance on a variety of platforms, the programmer also has the option to further optimize code for specific platforms without compromising its portability.

III. Framework Overview

Figure 1 provides a high level overview of the framework that is under development. This work describes the *Sycl based algorithm library*, which is highlighted in dashed box. The other components are currently in development in parallel projects.

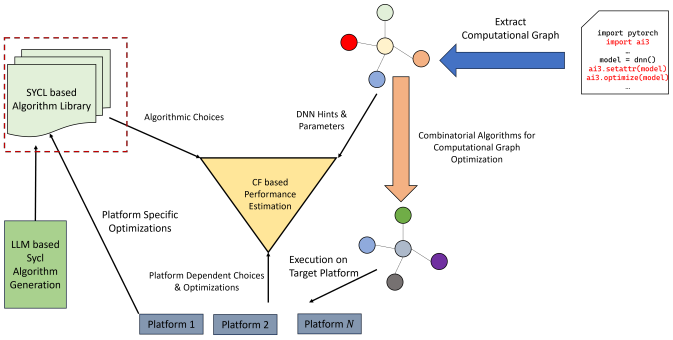

Fig. 1. Performance Portable DNN Framework

To provide context, we will briefly describe key components of the framework before delving deep into the SYCL based algorithm library.

User Interface: A user (CNN application developer) simply needs to import our library (titled `ai3` - Algorithmic Innovations for Accelerated Implementations of AI algorithms) and call two functions: (a) `setattr`: This optional function allows users to provide any additional hints to the backend framework such as latency or throughput requirements, accuracy guarantees, energy budget, maximum instantaneous power, etc. (b) `optimize`: This function extracts the computational graph of the model and calls the backend framework to produce an optimized computational graph for execution on the target platform.

SYCL based Algorithm Library: This library contains SYCL based implementations of a plethora of Convolution Algorithms that have been proposed by researchers. More details on this library are provided in Section IV

Collaborative Filtering (CF) based Performance Estimation: One of our concurrent projects focuses on developing a CF based performance estimation model (similar to [28]) that given a Convolution layer from the CNN model, an algorithm from the Algorithm Library, and a target platform (represented using devices parameters such as type, on-chip memory, memory bandwidth, compute capabilities), outputs the expected execution time, maximum instantaneous power, and energy consumption. Given the combinatorial nature of the problem, matrix completion based approach such as CF is being adopted for this problem.

Combinatorial Algorithm for Computational Graph Optimization: In our previous works [9], we modeled the problem of optimal layerwise selection of convolution algorithm for minimizing the execution time as a quadratic partitioned boolean integer program targeting FPGAs and on a limited set of convolution algorithms. One of our concurrent projects focuses on extending this algorithm to our sycl based framework, handling non series-parallel CNN graphs, and optimizing for multiple objectives: execution time, total energy, maximum instantaneous power.

LLM Based SYCL Algorithm Generation: Another dimension of productivity that our framework targets is that of incorporating newer convolution algorithms rapidly into the framework. Towards this goal, one of our concurrent projects is focused on developing an end to end pipeline that given a pseudo-code of a convolution algorithm from a research paper, produces a syntactically, functionally correct SYCL implementation that also correctly implements the parallelism parameters instructed by the application developer.

Inference vs Training: Our performance portable framework is currently limited to a single CPU-GPU or CPU-FPGA platform as we have not incorporated performance models and optimization formulations for multi-node systems. Thus, our framework is limited to inference, or training on a single node. In future, we plan to expand it to multiple nodes by extending the projects currently underway.

IV. SYCL based Algorithm Library

In this work, we target an Intel GPU and FPGA to develop a library of convolution algorithms. To achieve performance portability, we make the following two key decisions: (a) We use SYCL as the programming framework to implement our algorithms as it is supported on a variety of platforms from the leading vendors such as Intel, AMD, and NVIDIA. and (b) Whenever possible, we rely on the device optimized BLAS libraries instead of writing our own custom kernel IPs. As this work targets Intel devices, we use Intel MKL libraries for matrix multiplication implementation.

We have implemented the following convolution algorithms in our library as of the writing of this paper:

1) IM2COL [10]: One of the earliest algorithms that represented convolution layer operations as matrix multiplications by converting the reception under each stride

of the kernel as a single column with the kernel being converted as a single row which gets multiplied with the column. We implement this algorithm by transforming the input and kernel into matrix format and calling the gemm function of Intel MKL library.

2) KN2ROW [11]: An improvement over IM2COL algorithms that again represents convolution as matrix multiplication but avoids duplication of the input image. This is achieved by representing the $k \times k$ convolution as k^2 1×1 convolutions and performing shift and add to accumulate the result. We implement this algorithm by transforming the input and kernel into matrix format followed by a `parallel_for` to accumulate the partial results in parallel.
3) Depthwise Separable Convolution (DEPTHWISE) [6]: This version of the convolution algorithm was proposed to reduce the computation requirements of Convolution Layer to target edge devices. In this algorithm, instead of performing $C_{in} \times C_{out}$ convolutions with $k \times k$ filters, only C_{in} convolutions are performed with $k \times k$ filters followed by $C_{in} \times C_{out}$ convolutions with 1×1 filters leading to a k^2 reduction in computation. In our implementation, the C_{in} convolutions with $k \times k$ filters are performed similar to IM2COL algorithm while $C_{in} \times C_{out}$ convolutions with 1×1 filters are performed using KN2ROW algorithm.
4) DIRECT Convolution [29]: This version of convolution uses windowed operations as performed in the original convolution algorithm. Tiling is used to optimize the use of on-chip memory and improve cache utilization. We implement this algorithm using `parallel_for`.
5) Scalar Matrix Multiplication (SMM) [16]: This version of convolution avoids matrix multiplications and replaces it with matrix scaling operations. This is achieved by treating each output image as a linear combination of input images with kernel weights as the coefficients. Again, we implement this algorithm using `parallel_for`.

Future Work: We are implementing other convolution algorithms such as indirect convolution [21] and deformable convolution [12] to incorporate into the library. Moreover, we are also adding support for grouped convolution [30] by representing the relationship between the input and output channels as an adjacency matrix. Furthermore, we are also incorporating AMD's AOCL-BLAS [31] library into the implementation to support AMD GPU and AMD-Xilinx FPGAs.

V. Experimental Results

A. Setup

We implemented the 5 Convolution algorithms described in Section IV in SYCL/oneAPI, a framework designed for parallel computing across heterogeneous systems. We used 3 popular CNN models VGG16 [32], ResNet [33] and InceptionV4 [33] to conduct thorough experiments. In the current version of the framework, we simply plugged in our implementations of the convolution algorithms by loading the shared library.

We used Intel® Stratix® 10 FPGAs and Intel® Iris® Xe MAX GPUs available on Intel OneAPI Devcloud [34] to run our experiments and evaluate the execution times of the individual convolution layers as well as the total inference time per image.

Baselines: We use vanilla pytorch [35] and Intel extensions of pytorch [18] as baselines. Both these frameworks currently support only CPU and GPU, so we used their performance on GPUs as the baseline. Note that Intel extension of pytorch is an optimized version of pytorch, specifically for Intel devices. Thus, even modest improvements against this baseline (which we demonstrate in this paper), while providing portability to other platforms (our future direction) is a worthy goal.

B. Research Questions

Our goal in this paper was to answer the following three questions:

1) What is the relative performance of various convolution algorithms on the layers of the three selected CNN models on GPU versus FPGA platforms?
2) What is the relative performance of various convolution algorithms on the layers of the three selected CNN models compared against a pytorch and an Intel extensions for pytorch implementation?
3) What is the relative performance of the *Total Inference Time* of various convolution algorithms on the three selected CNN models on GPU and FPGA platforms when compared against a pytorch implementation and Intel extensions for pytorch implementation?

The first question addresses the benefits of heterogeneous computing platforms and answers questions on when one platform performs against the other, while the second question addresses the question on the merits of research on novel convolution algorithm development.

The third question highlights the additional overheads our proposed framework incurs in calling the shared library compared to pytorch or Intel extensions for pytorch, that have been optimized heavily and motivates our immediate future direction on engineering optimizations to reduce the overheads.

C. Target Platform Performance Comparison

For each CNN model, we analyze the performance of different convolution algorithms across the two platform GPU and FPGA. As noted before, the convolution algorithms are implemented in SYCL/OneAPI and called from a pytorch file that hosts the model. Additionally, we illustrate the results of the baseline pytorch and intel extensions for pytorch implementation on GPU for comparison. The numbers shown in brackets in the legends denote the total execution time of the convolution layers of the model, i.e., the sum of execution times of all the convolution layers.

VGG16: Figure 2 illustrates the results for VGG16. We draw the following general conclusions: (i) All the convolution algorithms on both GPU and FPGA show improvements

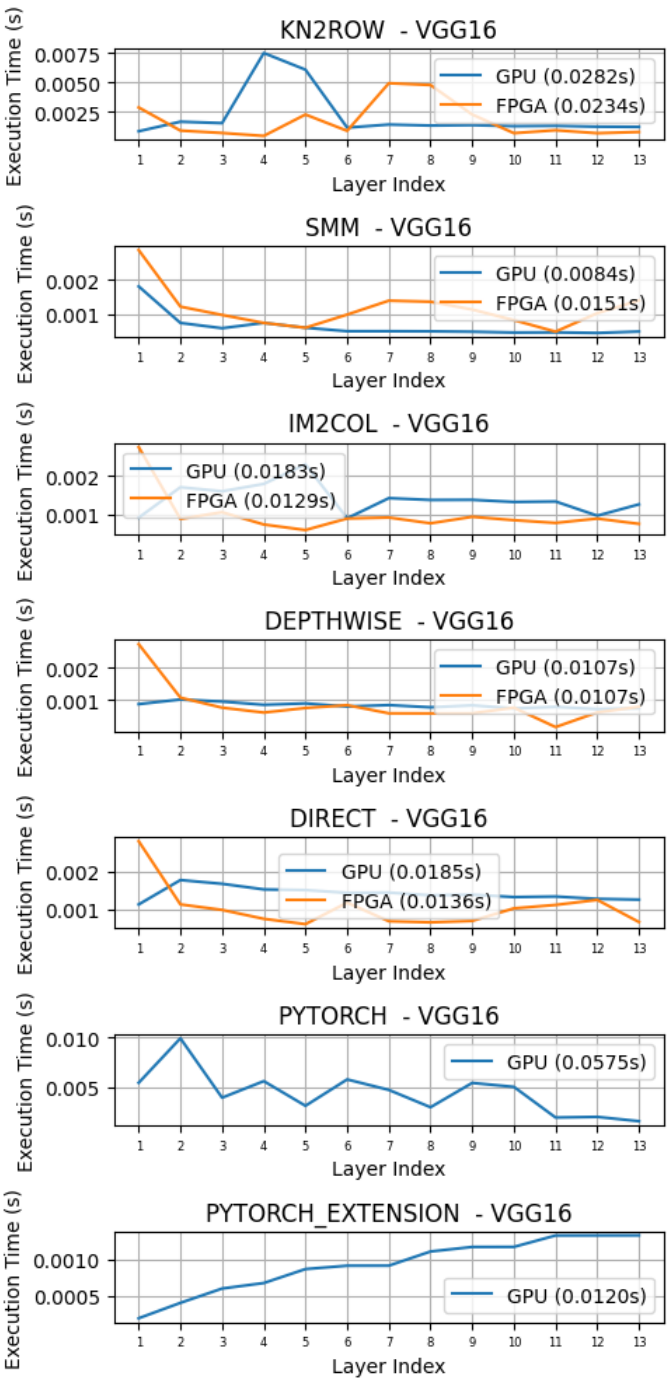

Fig. 2. Comparison of different platforms on VGG16

when compared to the pytorch implementation. The best Convolution performance for GPU is achieved using the SMM algorithm (7x improvement against pytorch) while for FPGA it is achieved using the DEPTHWISE algorithm (5.3x improvement against pytorch). However, as DEPTHWISE may impact the accuracy, the IM2COL (4.4x) can be considered as the best convolution algorithm on FPGA for VGG16 if functional correctness (compared to baseline pytorch) needs to be guaranteed. Against intel extensions for pytorch, SMM and Depthwise on GPU achieve performance improvements with SMM achieving the best performance improvement (1.5x). For FPGA, DEPTHWISE obtains higher performance than intel extensions for pytorch on GPU (while losing accuracy) while IM2COL achieves performance that is very close (0.93x speedup) to intel extensions for pytorch on GPU. (ii) For SMM convolution algorithm, GPU achieves a higher performance than FPGA (1.8x higher), while for IM2COL and DIRECT, FPGA achieves a higher performance than GPU (1.5x higher). They achieve similar performance for DEPTHWISE Convolution. (iii) Relative performance of GPU and FPGA also changes with layer dimensions. For VGG16, we do not observe a case where either GPU or FPGA dominates the other for all the layers for any given convolution algorithm.

Resnet: Figure 3 illustrates the results for the Resnet50 model. We draw the following general conclusions: (i) Again, all the convolution algorithms on both GPU and FPGA show improvements when compared to the pytorch implementation. The best Convolution performance for GPU is achieved using the SMM algorithm (7.7x improvement against pytorch) while for FPGA it is again achieved using the DEPTHWISE algorithm (4.1x improvement against pytorch). KN2ROW performs best for FPGA (4x improvement against pytorch) if functional correctness (compared to baseline pytorch) needs to be guaranteed. Against intel extensions for pytorch, only the SMM algorithm on GPU obtains a higher performance (1.4x speedup) and no algorithm on FPGA performs better. (ii) FPGA performs better than GPU for KN2ROW (1.3x better) and IM2COL (1.7x better), while GPU performs better than FPGA for SMM (4.5x better), DEPTHWISE (1.27x better), and DIRECT (2.4x better). In DIRECT convolution, apart from the initial layers, the performance of FPGA and GPU are similar. (iii) In this case, the relative performance of GPU and FPGA across the layers do not vary with the target device as they did in VGG16. FPGA performs better than GPU over all layers when using KN2ROW and IM2COL while GPU performs better for all the layers for SMM. DIRECT and DEPTHWISE do exhibit such a variation.

Inception-V4: Figure 4 illustrates the results for the Inception-V4 model. We show the results of every third layer due to the large number of layers in the model. However, the total convolution execution in the legends is computed by considering all the layers. For this model, we draw the following conclusions: (i) Again, all the convolution algorithms on both GPU and FPGA show improvements when compared to the pytorch implementation. The best Convolution performance for GPU is again achieved using the SMM algorithm (7.7x

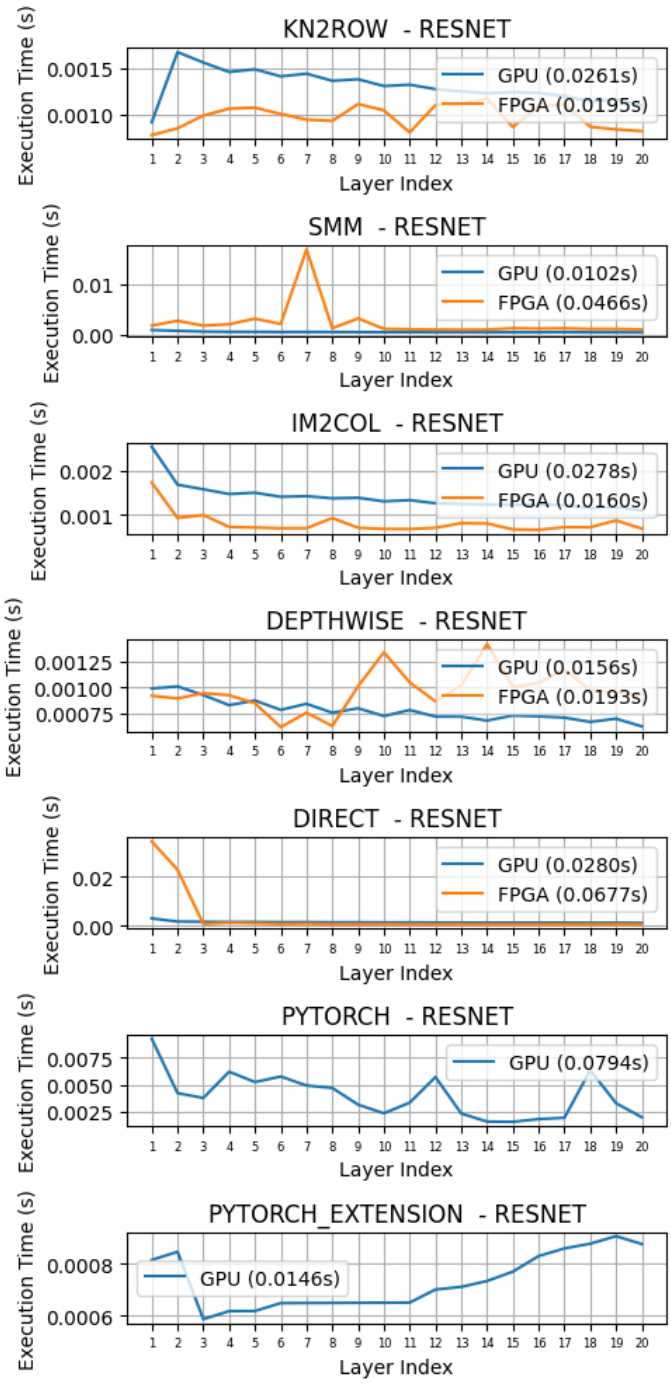

Fig. 3. Comparison of different platforms on Resnet

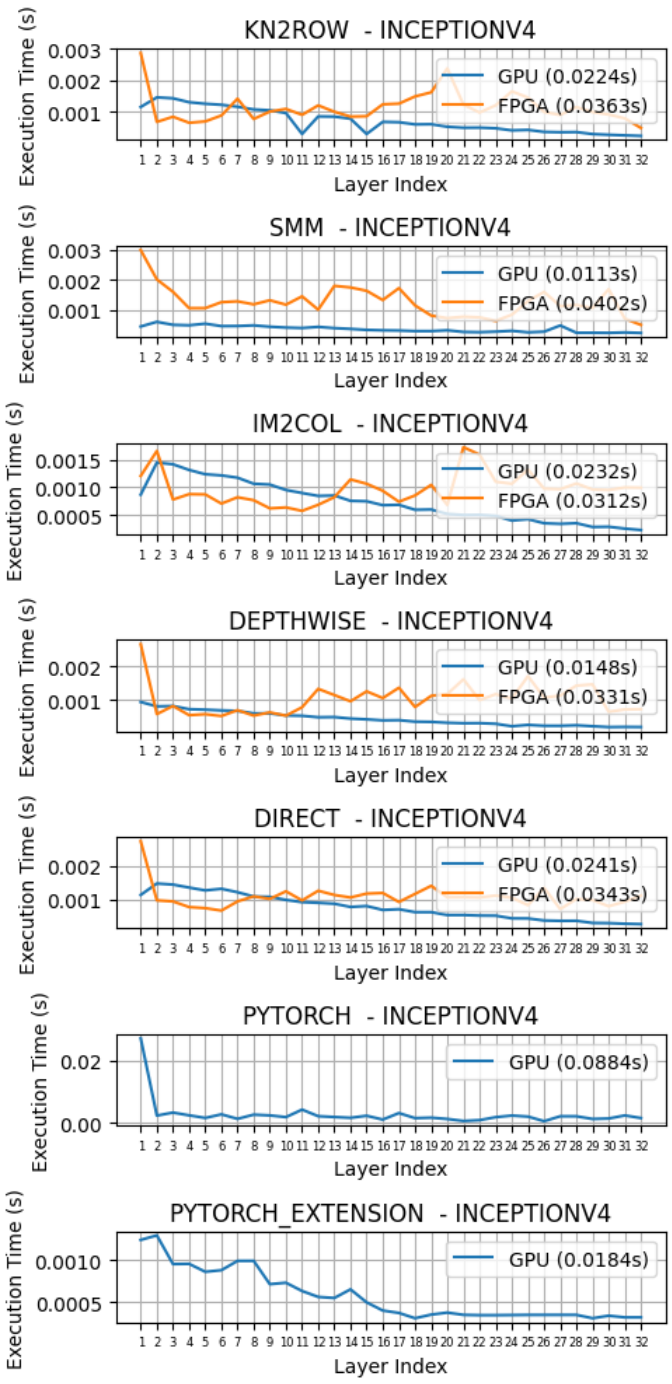

Fig. 4. Comparison of different platforms on Inception-V4

improvement over pytorch) while for FPGA it is achieved using the IM2COL algorithm (2.93x improvement over pytorch). Against intel extensions for pytorch, SMM and Depthwise on GPU achieve performance improvements with SMM achieving the best performance improvement (1.6x). None of the algorithms obtain a higher performance than intel extensions for pytorch on FPGA. (ii) In this model GPU always performs better than FPGA for all the algorithms. This maybe because the large model size leads to better utilization of the GPU resources. (iii) Even though GPU always performs better than FPGA for all the algorithms, their relative performance still exhibits variability across the layers.

D. Convolution Algorithm Performance Comparison

In this section, we discuss the results of the comparison of the relative performance of the convolution algorithms on the chosen target platforms. Again the number in bracket in the legends denote the total execution time of the convolution layers of the model. We also show the performance of the baselines pytorch and intel extensions for pytorch implementations on GPU for comparison.

GPU: Figure 5 illustrates the performance of the various convolution algorithms on the GPU platform. For VGG16, SMM achieves the best performance (3.5x speedup versus the lowest performing algorithm KN2ROW). For Resnet, again SMM achieves the best performance (2.74x speedup versus the lowest performing algorithm DIRECT). For Inception-V4, again SMM achieves the best performance (2.2x spedup versus the lowest performing algorithm, again DIRECT). The relative performance of algorithms also vary with layers, motivating for the need of switching algorithms similar to our prior work [9].

FPGA: Figure 6 illustrates the performance of the various convolution algorithms on the FPGA platform. For VGG16, DEPTHWISE achieves the best performance (2.15x speedup versus the lowest performing algorithm KN2ROW), while IM2COL achieves the best performance while guaranteeing functional correctness of the convolution layers (1.8x speedup versus the lowest performing algorithm KN2ROW). For Resnet, IM2COl achieves the best performance (4.2x speedup versus the lowest performing algorithm DIRECT). For Inception-V4, again IM2COl achieves the best performance (1.34x speedup versus the lowest performing algorithm SMM). Here again, we see a variability in the relative performance of the convolution algorithms across the layers.

E. Additional Overheads of our Framework

As mentioned before, we simply plugged in our convolution algorithms in pytorch implementations of the CNN by calling the shared library functions. Figure 7 illustrates the results. The labels pytorch extension and pytorch corresponds to the baseline implementations, while the total execution time for each CNN model using each convolution algorithm on GPU and FPGA are shown in relation to the intel extensions for pytorch implementation. As evident from Figure 7, the

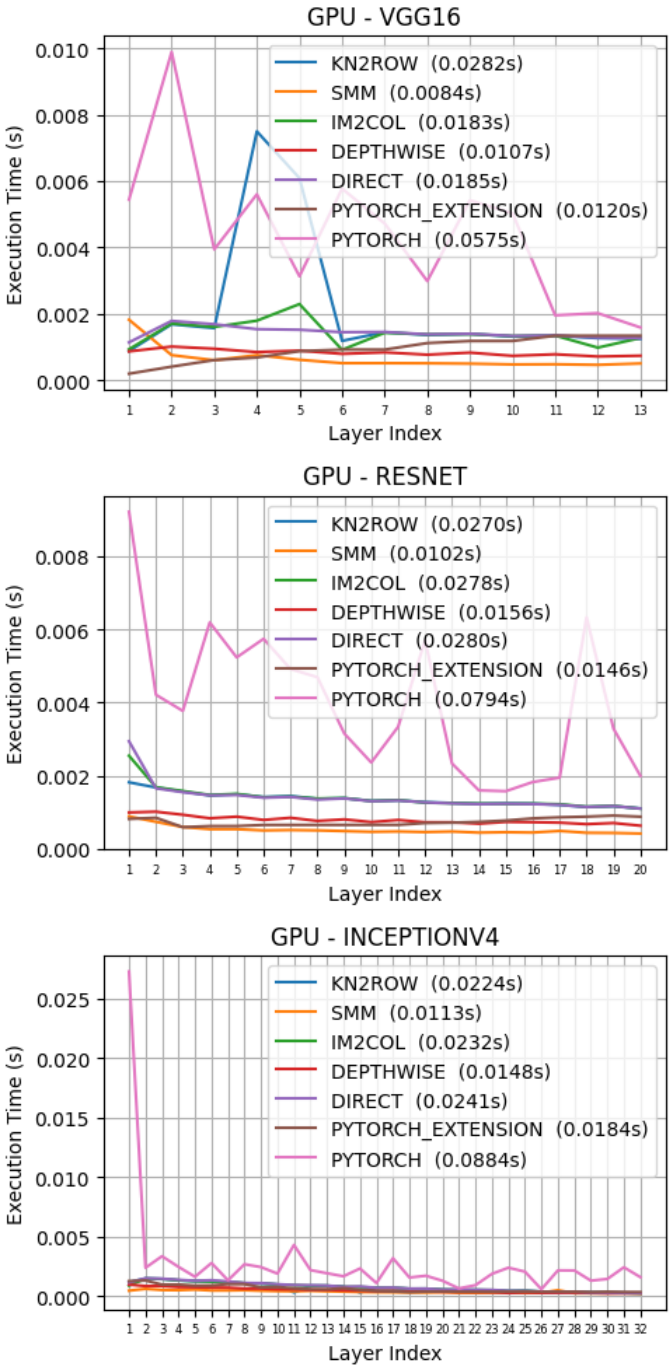

Fig. 5. Comparison of Convolutions on GPU across CNN Models

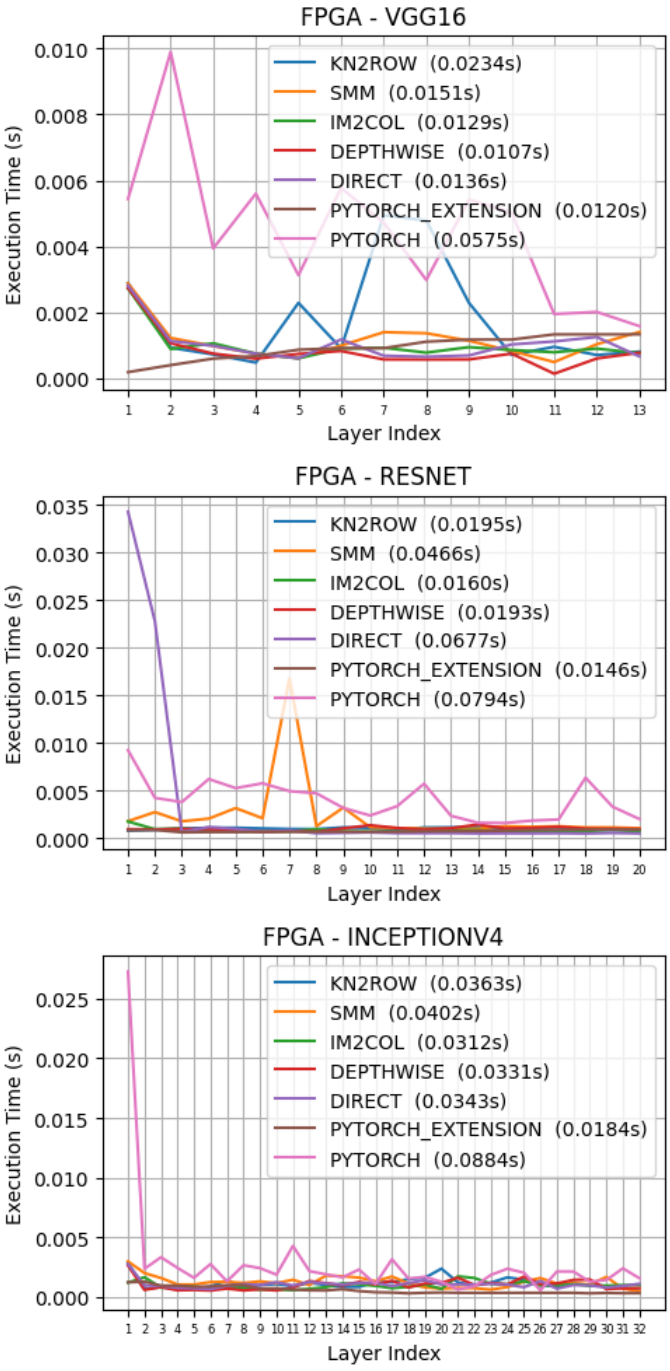

Fig. 6. Comparison of Convolutions on FPGA across CNN Models

performance on FPGA is 4.5 to 4.8 times slower while on GPU it is 1.6 to 2 times slower.

Thus, our immediate next step is to perform code profiling and optimizations to reduce the overheads of our framework to make it comparable to pytorch implementations.

VI. Acknowledgements

We acknowledge Intel for providing access to the Intel DevCloud platform for running the experiments. We also acknowledge other CWRU student researchers in Prof. Kuppannagari's research group who are working on the concurrent projects mentioned above and have provided invaluable feedback. These include Timothy Cronin, Nathaniel Tomczak, Wiam Skakri, Abhinav Khanna, and Yash Malhotra.

VII. Conclusion

In this work we presented our approach to develop a performance portable library of convolution algorithms using SYCL/oneAPI. We demonstrated improved convolution performance on both GPU and FPGA platforms compared to the baseline pytorch and Intel extensions for pytorch implementations. We also identified the limitations of our framework in terms of the additional overheads in calling our convolution implementations and plan to address them in near future.

Additionally, in future, we plan to integrate more devices such as NVIDIA and AMD GPUs as well as AMD-Xilinx's FPGAs into our framework. We anticipate that the choice of a portable language such as SYCL and the use of BLAS libraries as the computational kernels (instead of developing our own custom code/IPs) will enable us to rapidly expand the framework to the wide variety of platforms that are available and are in development to cater to the increasing need of AI/ML applications.

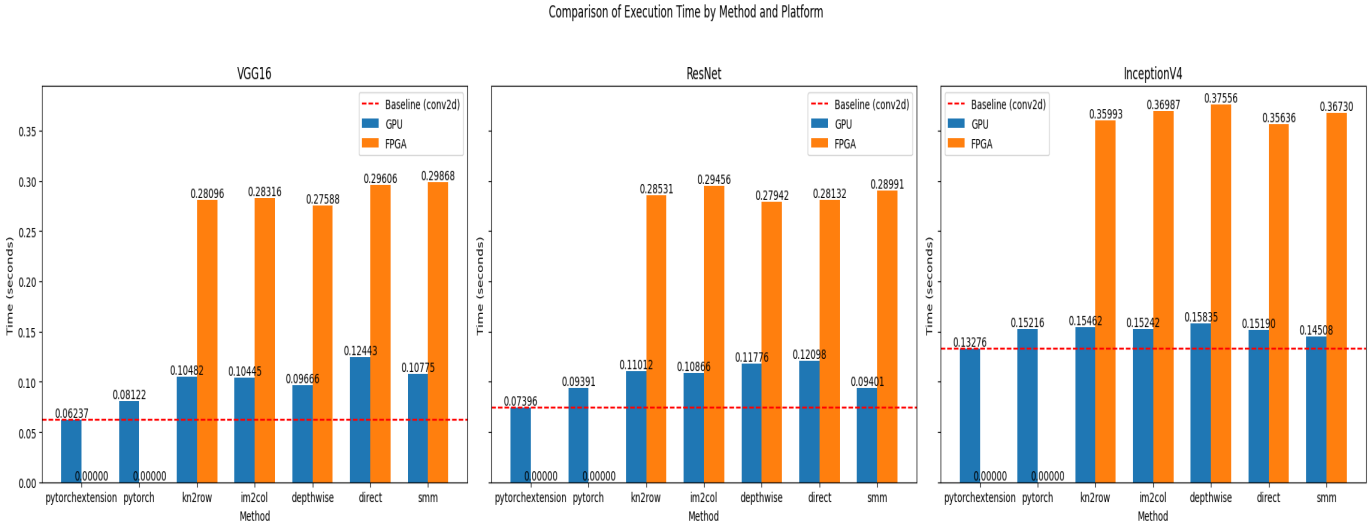

Fig. 7. Relative Execution Time (pytorch as Baseline)

REFERENCES

[1] H. O. Ikromovich and B. B. Mamatkulovich, "Facial recognition using transfer learning in the deep cnn," *Open Access Repository*, vol. 4, no. 3, pp. 502–507, 2023.

[2] S. M. Anwar, M. Majid, A. Qayyum, M. Awais, M. Alnowami, and M. K. Khan, "Medical image analysis using convolutional neural networks: a review," *Journal of medical systems*, vol. 42, pp. 1–13, 2018.

[3] T. Nakazawa and D. V. Kulkarni, "Wafer map defect pattern classification and image retrieval using convolutional neural network," *IEEE Transactions on Semiconductor Manufacturing*, vol. 31, no. 2, pp. 309–314, 2018.

[4] N. Tomczak and S. Kuppannagari, "Automated indexing of tem diffraction patterns using machine learning," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7, IEEE, 2023.

[5] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, *et al.*, "A survey on vision transformer," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 1, pp. 87–110, 2022.

[6] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

[7] N. Siddique, S. Paheding, C. P. Elkin, and V. Devabhaktuni, "U-net and its variants for medical image segmentation: A review of theory and applications," *Ieee Access*, vol. 9, pp. 82031–82057, 2021.

[8] W. Wang, J. Dai, Z. Chen, Z. Huang, Z. Li, X. Zhu, X. Hu, T. Lu, L. Lu, H. Li, *et al.*, "Internimage: Exploring large-scale vision foundation models with deformable convolutions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14408–14419, 2023.

[9] Y. Meng, S. Kuppannagari, R. Kannan, and V. Prasanna, "Dynamap: Dynamic algorithm mapping framework for low latency cnn inference," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '21, (New York, NY, USA), p. 183–193, Association for Computing Machinery, 2021.

[10] S. Chetlur, C. Woolley, P. Vandermersch, J. M. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *ArXiv*, vol. abs/1410.0759, 2014.

[11] A. Anderson, A. Vasudevan, C. Keane, and D. Gregg, "High-performance low-memory lowering: Gemm-based algorithms for dnn convolution," in *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 99–106, 2020.

[12] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 764–773, 2017.

[13] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Lincoln ai computing survey (laics) update," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7, IEEE, 2023.

[14] G. Singh, A. Wagle, S. Khatri, and S. Vrudhula, "Cidan-xe: Computing in dram with artificial neurons," *Frontiers in Electronics*, vol. 3, p. 834146, 2022.

[15] M. La and A. Chien, "Cerebras systems: Journey to the wafer-scale engine," *University of Chicago, Tech. Rep*, 2020.

[16] O. Amir and G. Ben-Artzi, "Smm-conv: Scalar matrix multiplication with zero packing for accelerated convolution," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3066–3074, 2022.

[17] S. Yi, J. Ju, M.-K. Yoon, and J. Choi, "Grouped convolutional neural networks for multivariate time series," *arXiv preprint arXiv:1703.09938*, 2017.

[18] Intel, "Pytorch optimizations from intel." https://www.intel.com/content/www/us/en/developer/tools/oneapi/optimization-for-pytorch.html, 2023. Accessed:2024-01-28.

[19] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[20] J. Zhang, F. Franchetti, and T. M. Low, "High performance zero-memory overhead direct convolutions," in *International Conference on Machine Learning*, pp. 5776–5785, PMLR, 2018.

[21] M. Dukhan, "The indirect convolution algorithm," *arXiv preprint arXiv:1907.02129*, 2019.

[22] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating cnn inference on fpgas: A survey," *arXiv preprint arXiv:1806.01683*, 2018.

[23] G. Habib and S. Qureshi, "Optimization and acceleration of convolutional neural networks: A survey," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 7, pp. 4244–4268, 2022.

[24] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating cnn inference on asics: A survey," *Journal of Systems Architecture*, vol. 113, p. 101887, 2021.

[25] T. Pacini, E. Rapuano, and L. Fanucci, "Fpg-ai: A technology-independent framework for the automation of cnn deployment on fpgas," *IEEE Access*, vol. 11, pp. 32759–32775, 2023.

[26] M. W. Moskewicz, F. N. Iandola, and K. Keutzer, "Boda-rtc: Productive generation of portable, efficient code for convolutional neural networks on mobile computing platforms," in *2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 1–10, IEEE, 2016.

[27] B. Johnston, J. S. Vetter, and J. Milthorpe, "Evaluating the performance and portability of contemporary sycl implementations," in *2020 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pp. 45–56, 2020.

[28] S. Salaria, A. Drozd, A. Podobas, and S. Matsuoka, "Predicting performance using collaborative filtering," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 504–514, IEEE, 2018.

[29] J. Zhang, F. Franchetti, and T. M. Low, "High performance zero-memory overhead direct convolutions," *ArXiv*, vol. abs/1809.10170, 2018.

[30] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.

[31] AMD, "Dense linear algebra: Aocl-blas and aocl-lapack." https://www.amd.com/en/developer/aocl/dense.html, 2023. Accessed:2024-01-28.

[32] S. Mascarenhas and M. Agarwal, "A comparison between vgg16, vgg19 and resnet50 architecture frameworks for image classification," in *2021 International conference on disruptive technologies for multi-disciplinary research and applications (CENTCON)*, vol. 1, pp. 96–99, IEEE, 2021.

[33] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017.

[34] Intel, "Intel devcloud." https://devcloud.intel.com/oneapi/get_started/, 2023. Accessed:2024-01-28.

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.