Achieving enhanced inference using automated optimization of deep learning compilers

Shreya Vairava Subramanian
sv523@cam.ac.uk
Department of Computer Science and Technology
University of Cambridge
United Kingdom

Abstract

Tools developed to achieve faster training and inference are deep learning compilers. Deep learning models often face the danger of maintaining efficiency. This could be due to issues such as higher computational cost, dependency on advanced hardware technologies, scalability, memory consumption, energy consumption and more. In order to overcome these challenges, automated optimization strategies can be implemented to achieve higher efficiency of models during training and inference. The main goal of this paper is to analyze how such strategies can be applied to these complex models to maintain efficiency without succumbing to such challenges. Different types of automated optimization techniques were trained on popular datasets to analyze the best combination required to achieve faster efficiency.

1 Introduction

A subset of artificial intelligence to model complex patterns is deep learning [1]. Deep learning compilers [2] are defined as tools that convert high - level code such as those written using tensorflow to low - level code such as those that performs operations to speed up the process of training and inference. There are different types of such compilers including TensorRT [3], Tensor Virtual Machine (TVM) [4], Accelerated Linear Algebra (XLA), Glow (Facebook) [5], Apache MXNET Compiler [6]. As models grow with respect to size and complexity, it becomes resource - intensive to code such models which may potentially include significant number of parameters, extensive cost for computation and larger usage of memory. In order to address these challenges, automated optimization replaces the burden of manual optimization. Automated optimization techniques such as Operator Fusion [7], Quantization [8], Pruning [9], Memory Optimization [10], Layer Fusion [11], Automatic Mixed Precision [12], Batching [13] and Parallelization [14] are incorporated into popular datasets like MNIST [16], CIFAR - 10 [17], CIFAR - 100 [18], SVHN [19] and FASHION - MNIST [20] with the aim to achieve enhanced efficiency during training.

2 Literature Review

With the five different types of deep learning compilers mentioned above, TensorRT, developed by NVIDIA was considered to be worked upon due to it's record of achieving best performance on hardware accelerators like Graphical processing Units [21] and Tensor Processing Units [22]. Another advantage was the ability to support mixed - precision computation making it ideal to achieve faster outcomes with less delay. TVM was found to be heavily complicated in terms of setup and integration to achieve better efficiency [23]. XLA was found to be less hardware - agnostic i.e., the ability to work across types of hardware platforms in comparison to TensorRT [24]. Glow was specifically designed for mobile application which lacks the power of advanced accelerators like GPUs / TPUs [25]. MXNET was found to be working well only within the Apache ecosystem and lacked scalability to perform in other environments [26].

3 Methodology

The first step is to identify the datasets to be worked upon in order to understand the scope of performance. Popular datasets like the ones mentioned before were taken for implementation. Figure 1 shows the compiled list of dataset.

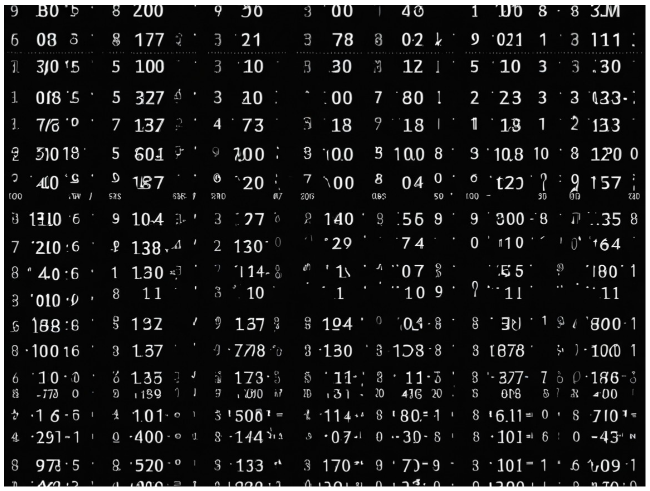

Figure 1: Dataset

The architecture implemented for these tasks are as follows.

(1) A CNN architecture was implemented for MNIST.
(2) A ResNet - 18 architecture was implemented for CIFAR - 10 and CIFAR - 100.
(3) A VGG - 16 architecture was implemented for FASHION - MNIST.
(4) An EfficientNet architecture was implemented for SVHN.

3.1 Operator Fusion

Operator Fusion is a technique in which the number of operations performed during automated optimization is reduced by concatenating multiple operations into a single operation. In a deep learning model, there might be several operations a neural network has to undergo. This potentially includes Linear operations such as matrix addition [27], matrix multiplication [28] , convolution [29] Non - Linear operations such as activation functions [30] Normalization operations including batch normalization [31], layer normalization [32], instance normalization [33] Pooling operations [34] Regularization operations including dropout rate [35] and regularization strategies [36] Loss functions [37]

The advantage of using operator fusion is to reduce memory consumption and fasten the computational procedure.

3.2 Quantization

Quantization is defined by decreasing the precision. The 32 - bit floating point numbers are converted to 16 - bit floating point numbers and 8 - bit floating point numbers.

There are different types of quantization.

(1) ***Quantization - aware training*** - It is performed for obtaining lower precision numbers.
(2) ***Binary and Ternary Quantization*** - These are advanced quantization methods used to reduce precision to -1 and +1 for binary operation and -1, 0, +1 for ternary operations.

The advantage of using quantization for optimization is to reduce the size of the model and increase the consistency during inference.

3.3 Pruning

(1) ***Weight Pruning*** - The size of the model can be reduced by weight pruning which eliminates the share of weights based on their significance in the neural network.
(2) ***Channel Pruning*** - The inference can be fastened by removing a set of neurons from the neural network.
(3) ***Structured Pruning*** - The process of removing a set of parameters including neurons, weights, activation functions etc. by considering the structure of the neural network.
(4) ***Unstructured Pruning*** - The process of removing individual parameters including neurons, weights, activation functions etc. without considering the structure of the neural network.

3.4 Memory Optimization

The process of reducing the memory capacity while performing training and inference on a model is called memory optimization. This works by reusing the memory over time. A block of memory is reused when it's previous requirement is finished. The concept of buffer sharing where memory blocks are dynamically assigned to various jobs during training and inference. Some of the memory optimization strategies include :

(1) ***In - Place Operations*** - The need for extra memory allocation can be reduced by modifying existing tensors instead of creating new ones.
(2) ***Tensor Recycling*** - The unused memory of existing tensors can be allocated to new tensors.
(3) ***Dynamic Memory Management*** - Memory can be dynamically allocated during execution through TensorRT.

3.5 Layer Fusion

Similar to how operator fusion works by concatenating multiple operations, layer fusion works by concatenating multiple layers in a neural network. The advantage of using layer fusion is to reduce computational overhead and memory bandwidth.

3.6 Automatic Mixed Precision

Different to quantization, automatic mixed precision is used to decrease the precision but by combining multiple formats. Formats like FP16, used for arithmetic operations and FP32, used for scientific operations are combined in this procedure. The different types of automatic mixed precision are :

(1) ***Static Mixed Precision*** - The type of operation is manually defined in this procedure on whether a lower precision or a full precision have to employed.
(2) ***Dynamic Mixed Precision*** - Instead of manually defining the operation, the precision is mixed dynamically during execution.
(3) ***Layer - Wise Mixed Precision*** - Different types of precision values are assigned to different layers in a neural network.

3.7 Batching & Parallelization

Similar to how batch normalization is employed in neural networks, batching is the process of splitting the dataset into a certain group to simplify the training and inference by not overcrowding the memory. The power of modern hardware accelerators can be leveraged by batching the data into smaller groups. The optimization can become efficient when data is processed in batches as memory access patterns become more accessible. The computational overhead can be reduced during batching. This is because the time spent on transferring the data between CPU and GPU can be minimized as there is less amount of memory consumed due to batching the data for processing.

4 References

LeCun, Yann, Bengio, Yoshua, & Hinton, Geoffrey. (2015). Deep learning. Nature, 521(7553), 436-444.

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Chen, Jian, Davis, Andy, Dean, Jeffrey, Devin, Matthias, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, et al. (2016). TensorFlow: A system for large-scale machine learning. OSDI.

Fast, Tony, Ochoa, Sebastian, & Saito, Shohei. (2020). TensorRT: High-performance inference on NVIDIA GPUs. NVIDIA Corporation.

Chen, Tianqi, Moreau, Trevor, Nguyen, Hieu, Zhang, Shuo, Bhat, Megha, Chen, Yujia, Rastegari, Mohammad, et al. (2018). TVM: An automated end-to-end optimizing compiler for deep learning. Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18).

Jouppi, Norman P., et al. (2017). In-Datacenter Performance Analysis of a Tensor Processing Unit. ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA).

Berg, Nikolai, Cichocki, Andrzej, et al. (2018). Glow: A Compiler for Deep Learning. Facebook Research.

Chen, Tianqi, et al. (2015). MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. arXiv:1512.01274.

Sze, Vivian, et al. (2017). Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE, 105(12), 2295-2329.

Jacob, Benoit, et al. (2018). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. arXiv:1712.05877.

Han, Song, Mao, Huizi, & Dally, William J. (2015). Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. ICLR.

Mundra, P., et al. (2019). Memory-efficient deep learning for large-scale models. Proceedings of the IEEE International Conference on Computer Vision.

Sze, Vivian, et al. (2017). Efficient Processing of Deep Neural Networks: A Tutorial and Survey. Proceedings of the IEEE, 105(12), 2295-2329.

Micikevicius, Petr, et al. (2018). Mixed Precision Training. International Conference on Learning Representations (ICLR).

Ba, Jimmy, et al. (2016). A Study on Overfitting in Deep Neural Networks. arXiv:1604.07117.

Dean, Jeffrey, et al. (2012). Large Scale Distributed Deep Networks. Advances in Neural Information Processing Systems (NIPS).

LeCun, Yann, et al. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, 86(11), 2278-2324.

Krizhevsky, Alex, et al. (2009). Learning Multiple Layers of Features from Tiny Images. Technical Report, University of Toronto.

Netzer, Y., et al. (2011). Reading Digits in Natural Images with Unsupervised Feature Learning. NIPS Workshop on Deep Learning and Unsupervised Feature Learning.

Xiao, Han, et al. (2017). Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv: 1708.07747.

NVIDIA. (2021). NVIDIA GeForce GTX 1080 Ti Graphics Card. NVIDIA Corporation.

Jouppi, Norman P., et al. (2017). In-Datacenter Performance Analysis of a Tensor Processing Unit. ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA).

Chen, Tianqi, et al. (2018). TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. OSDI 18.

Saito, Shohei, et al. (2020). TensorRT: High-performance inference on NVIDIA GPUs. NVIDIA Corporation.

Berg, Nikolai, et al. (2018). Glow: A Compiler for Deep Learning. Facebook Research.

Chen, Tianqi, et al. (2015). MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. arXiv:1512.01274.

Strang, Gilbert. (2009). Introduction to Linear Algebra. Wellesley -Cambridge Press.

Goodfellow, Ian, et al. (2016). Deep Learning. MIT Press.

Ioffe, Sergey, & Szegedy, Christian. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ICML.

Ba, Jimmy, Kiros, Ryan, & Hinton, Geoffrey. (2016). Layer Normalization. arXiv:1607.06450.

Ulyanov, Dmitry, et al. (2016). Instance Normalization: The Missing Ingredient for Fast Stylization. arXiv:1607.08022.

LeCun, Yann, et al. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, 86(11), 2278-2324.

Srivastava, Nitish, et al. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR.

Ng, Andrew Y. (2004). Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance. Proceedings of the Twenty-First International Conference on Machine Learning.