The International Journal of Engineering and Science (IJES)
|| Volume || 8 || Issue || 4 Series I || Pages || PP 39-44 || 2019 ||
ISSN (e): 2319 – 1813 ISSN (p): 23-19 – 1805

Low Precision Arithmetic Operations in Deep Neural Networks: An Overview

Anwarul Azim
Assistant Professor, Department of EEE, University of Asia Pacific Bangladesh

----------ABSTRACT----------
The main objective of this paper is to provide a comprehensive review on available methodologies and computational frameworks for efficient low precision arithmetic operations in Deep Neural Networks (DNN). Most widely used and popular DNN models use full precision (32 bit) of floating point operations for training and inference workloads. But recent research developments have demonstrated the capability of lower numerical precision operations with minimal or no loss of accuracy while reducing computational workload for servers and mobile devices.

KEYWORDS; -*Deep Neural Network (DNN), low precision, Computational Precision, Kernel API, Computational Primitives, operator fusion*

Date of Submission: 26-04-2019 Date of acceptance: 06-05-2019

I. INTRODUCTION

Deep Neural Networks (DNN) are increasingly becoming the popular solution for image classification and language modeling due to high inference accuracy. Larger DNN models offer better performance, but also results in bigger computational overhead and memory requirements. Almost all major technology companies such as Google (Tensorflow), Facebook (Pytorch), Apache (MXNET), Microsoft, Intel, Nvidia, Qualcomm etc. are providing open source deep learning frameworks and kernel libraries for developers and researchers. So far significant breakthroughs have been demonstrated in computer vision, natural language processing, artificial intelligence and other relevant areas.

Most commercial DNN applications available today use 32-bit floating point precision for training and inference workloads. A number of recent research projects have shown minimal or no loss of accuracy using low precision computation for training (16 bit) and inference (8 bit). Several hardware providers are now making math kernel libraries available to implement low numerical precision in DNN computations such as mixed precision training [1] from Nvidia, MKL-DNN [2] from Intel etc. This paper will give an overview of open source performance libraries available currently to implement low precision numerical operations for DNN training and inference in associated hardware architectures from various providers such as Nvidia, Intel, ARM, Qualcomm etc.

II. RELATED WORK

Deep learning training with 16-bit multipliers and inference with 8-bit multipliers or less of numerical precision have been demonstrated by researchers, accumulating to higher precision with minimal to no loss in accuracy across various models. Models trained by Courbariaux, et al. [3] on the MNIST, CIFAR-10, and SVHN datasets with lower numerical precision multipliers and high precision accumulators and updated the high precision weights; authors proposed combining dynamic fixed point (having one shared exponent for a tensor or high dimensional array) with Gupta, et al. [4] stochastic rounding as future work. Activations and weights are quantized by Vanhoucke, et al. [5] to 8-bits and kept the biases and first layer input at full precision (64 bit) for the task of speech recognition on CPUs. A simple network is trained by Hwang, et al. [6] with quantized weights of -1, 0 and 1 in the feed forward propagation and updated the high precision weights in the back propagation using the MNIST and TIMIT datasets with negligible performance loss. Weights and activations are encoded by Miyashita, et al. [7] in a base-2 logarithmic representation and trained in CIFAR-10 dataset with 5-bits weights and 4-bit activations resulting in minimal performance degradation. Rastegari, et al. [8] trained AlexNet with binary weights (except for the first and last layers) and updated on full precision weights with a top-1 2.9% accuracy loss. Based on their experiments, authors recommend avoiding binarization in fully connected layers and convolutional layers with small channels or filter sizes (e.g., 1x1 kernels). In the paper by Mellempudi, et al. [9], ResNet-101 is trained with 4-bit weights and 8-bit activations in convolutional layers while doing updates in full precision with a top-1 2% accuracy loss. Micikevicius, et al. [10] trained with 16-bit floating-point multipliers and full precision accumulators and updated the full precision weights with negligible to no loss in accuracy for AlexNet, VGG-D, GoogLeNet, ResNet-50, Faster R-CNN, Multibox SSD, DeepSpeech2, Sequence-

to-Sequence, bigLSTM, and DCGAN. Researchers at Baidu [11] successfully used 8-bits of fixed precision with 1 sign bit, 4-bits for the integer part and 3-bits for the fractional part.

Several quantization techniques are used by Sze, et al. [12] to show minimal to no loss at reduced precision (except for the first and last layers which were kept at full precision). Das, et al. [13] trained ResNet-50, GoogleNet, VGG-16, and AlexNet using 16-bits integer multipliers and 32-bit accumulators. Google researchers [14] described the bfloat16 numerical format that is natively supported in Google's Tensorflow Processing Unit (TPU) for training workloads, and will be available in future Intel Xeon processors and Intel Nervana Neural Processors.

III. OBJECTIVES

Performance of deep neural network training and inference is usually limited by numerical computation ability, memory budget or operational latency. Using reduced precision representation and computation, larger models can lower the operational overhead. Memory requirements are reduced by using fewer bits to store same number of values. Arithmetic operations can be made more efficient by ensuring higher throughput for low precision math. For example, recent GPUs from Nvidia can produce 2X to 8X higher throughput by using half precision (8 bit) calculations over single precision.

The goal of this paper is to present recent developments in DNN training and inference in reduced precision while maintaining model accuracy for inference tasks such image classifications. The most common 16-bit numerical formats and 8-bit numerical formats, respectively, are 16-bit IEEE floating point (*fp16*), bfloat16 (*bf16*), 16-bit integer (*int16*), 8-bit integer (*int8*), and 8-bit Microsoft floating point (*ms-fp8*).

IV. OVERVIEW OF LOW PRECISION LIBRARIES IN DIFFERENT HARDWARE ARCHITECTURES

For the mixed precision training using Nvidia GPUs, all tensors and arithmetic for forward and backward passes are FP16 [1]. So, weights, activations and gradients are stored as 16 bits, halving the requirements of FP32 training. Arithmetic operations for DNN training can be divided into three categories –vector dot products, reductions and point-wise operations. Some DNN models require that FP16 vector dot product accumulates the partial products into an FP32 value, which is converted into FP32 value before writing into memory. Otherwise, Fp16 models show reduced accuracy compared to FP32 models. Large reductions (sums across elements of a vector) usually performed in FP32 at batch normalization and softmax layers. In Nvidia implementation, tensor read and write operations are still 16 bits, but arithmetic is in FP32. For pointwise operations, either FP16 or FP32 math is used. Figure 2 shows the mixed precision training iteration for a layer:

Nvidia has provided a platform called TensorRT [15] built on CUDA (parallel programming model) for high performance deep learning low precision inference. TensorRT provides INT8 and FP16 optimizations for production deployments of DNN applications in video streaming, speech recognition, recommendation and natural language processing. Reduced precision inference significantly reduces application latency, which is a requirement for many real-time services, auto and embedded applications. Trained DNN models from every deep learning framework can be imported into TensorRT. After applying optimizations, TensorRT selects platform specific kernels to maximize performance on Tesla GPUs in the data center, Jetson embedded platforms, and NVIDIA DRIVE autonomous driving platforms.

For Intel architecture, Math kernel library for Deep Neural Networks (MKL-DNN) [16] is an open source performance library for Deep Learning (DL) applications intended for acceleration of DL. Intel MKL-DNN includes highly vectorized and threaded building blocks for implementation of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) with C and C++ interfaces. The library provides optimized implementations for the most common computational functions (primitives) used in deep neural networks covering a wide range of applications, including image recognition, object detection, semantic segmentation, neural machine translation, and speech recognition. The following table 1 shows supported functionalities with their variants:

Type	Primitives	fp32 Training	fp32 Inference	int8 Inference
Convolution	1D Direct Convolution	X	X	
	2D Direct Convolution	X	X	X
	2D Direct Deconvolution	X	X	X
	2D Winograd Convolution	X	X	X
	3D Direct Convolution	X	X	
	3D Direct Deconvolution	X	X	
Inner Product	2D Inner Product	X	X	X
	3D Inner Product	X	X	

RNN	LSTM	X	X	X
Activation	ReLU	X	X	X
	Sqrt	X	X	
	Abs	X	X	
	Linear	X	X	
	Softmax	X	X	
Data manipulation	Quantization	X	X	X
	Sum	X	X	X
	Concatenation	X	X	X
	Shuffle	X	X	X

Table 1: MKL-DNN supported functionalities for FP32 training and INT8 inference

MKL-DNN libraries have implemented for widely used DNN frameworks such as Caffe, Chainer, DeepBench, PaddlePaddle, Pytorch, Tensorflow, Microsoft Cognitive Toolkit (CNTK), Apache MXNet, OpenVino Toolkit, BigDL etc. that supports processors compatible with Intel 64 architectures.

Two recent feature developments fused computation and reduced-precision kernels have shown significant improvement in CPU based inference in MXNET framework using subgraph. During step 1 of MXNET graph optimization, subgraph would apply graph partitioning and all MKL-DNN operators will group into a node.

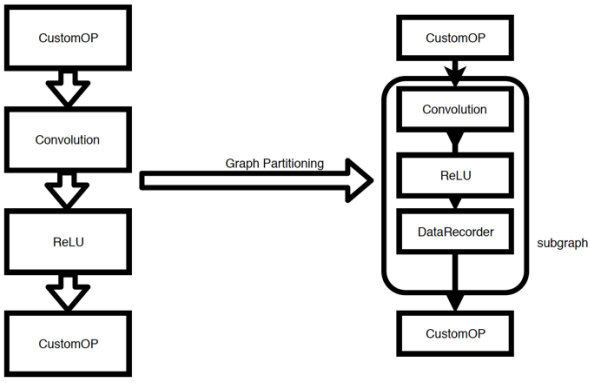

Figure 1: Symbolic mode in MKL-DNN optimization

Data format will adjust to either MKL-DNN internal format or NDarray format on subgraph boundary, depending on executing mode.For symbolic mode in figure 1, subgraph will try to cover adjacent MKL-DNN operators as much as possible.

On the other hand, each operator for imperative mode in figure 2 will execute in an independent subgraph, protecting MKL-DNN format from outside.

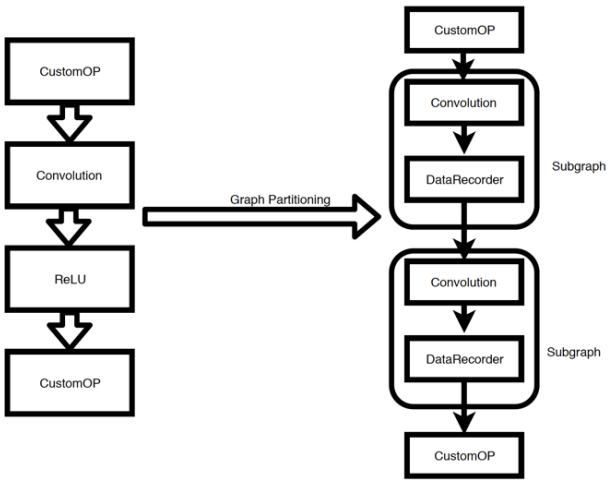

Figure 2: imperative mode in MKL-DNN optimization

In MKL-DNN library, new subgraphs that run multiple fused operators in single execution, can be generated to perform defined operations. The subgraph selector can create MKL-DNN convolutions such as convolution+relu or conv+batchnorm+relu to represent standalone operations for the library.

Finally, INT8 inference can be performed by MKL-DNN in MXNET, using fused primitives by generating INT8 model based on FP32 model and running QuantizeGraph pass inside subgraph to replace FP32 operators. As a result, inference can be done on new input data at most reduced precision. In various DNN topologies, INT8 inference provide a significant performance improvement in MXNET as shown in figure 3:

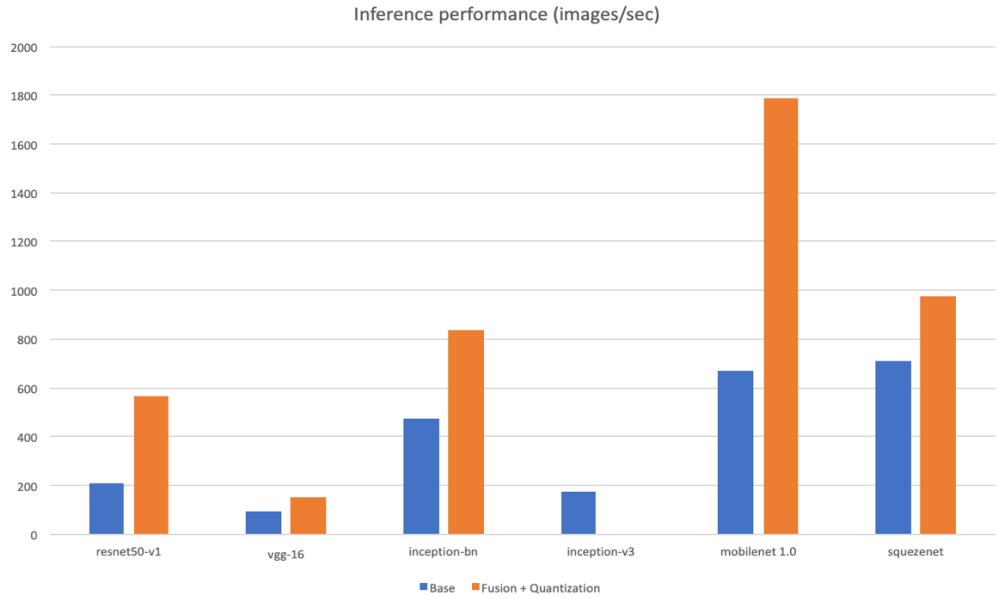

Figure 3: Performance Comparison for INT8 inference

In case of ARM architecture, low precision operations are implemented by Cortex Microcontroller Software Interface Standard (CMSIS) [17], a vendor-independent hardware abstraction layer for the Cortex-M processor series and defines generic tool interfaces. The CMSIS NN library enables consistent device support and simple software interfaces to the processor and the peripherals, simplifying software re-use, reducing the learning curve for microcontroller developers, and reducing the time to market for new devices. CMSIS-DSP (Digital Signal Processing) is an important component in this library that provides a DSP library collection with more than 60 functions for various data types: fixed-point (fractional q7, q15, q31) and single precision floating-point (32-bit). The library is available for all Cortex-M cores. Implementations that are optimized for the SIMD instruction set are available for Cortex-M4, Cortex-M7, Cortex-M33 and Cortex-M35P. The library is divided into a number of functions such as basic math functions, fast math functions, complex math functions, filters, matrix functions, transforms, motor control functions, statistical functions, support functions and interpolation functions. The library also has separate functions for operating on 8-bit integers, 16-bit integers, 32-bit integer and 32-bit floating-point values.

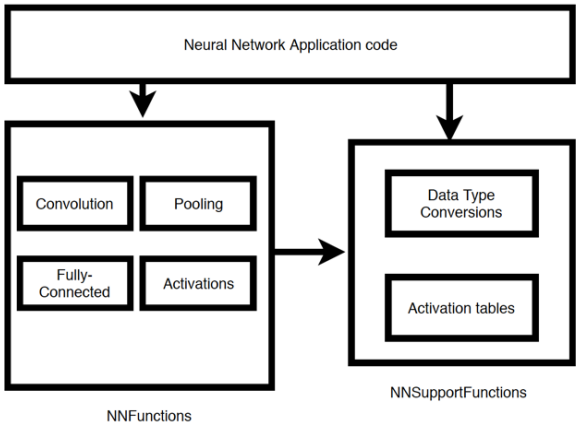

Figure 4: CMSIS-NN Library

CMSIS-NN library consists of two parts as shown in figure 4, namely NNSupportFunctions and NNFunctions. NNSupportFunctions consists of different utility functions, such as data conversion and activation function tables, which are used in NNFunctions. These utility functions can also be used by the application code to construct more complex NN modules, for example Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU).NNFunctions include the functions that implement popular neural network layer types, such as convolution, depthwise separable convolution, fully-connected (inner-product), pooling and activation. These functions are used by the application code to implement the neural network inference applications. The kernel APIs are also kept simple, so that it can be easily retargeted for any machine learning framework. In implementation for Convolutional Neural network (CNN), Vgg-16 and MobileNet DNN topologies in imagenet dataset and Tensorflow, INT8 inference accuracies show relatively better performance for smaller networks in size than larger ones, as shown in figure 5 below:

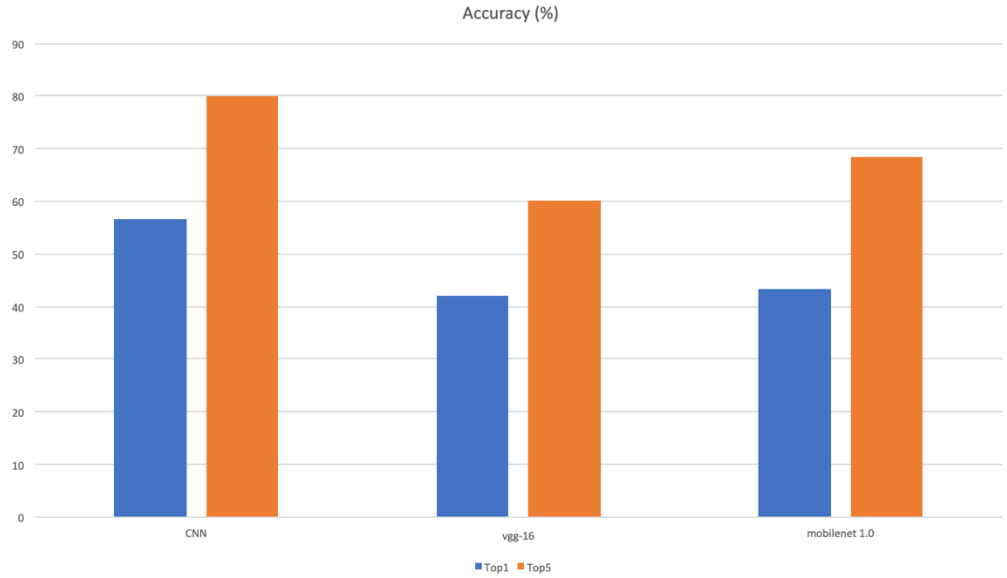

Figure 5: INT8 accuracy using CMSIS-NN

For high end mobile android devices, Qualcomm Snapdragon platforms can run trained DNN models locally for classification tasks. In this architecture, The Snapdragon Neural Processing Engine (SNPE) is a Qualcomm Snapdragon software accelerated runtime for the execution of deep neural networks. SNPE provides functionalities to convert caffe, caffe2, ONNX and Tensorflow models to SNPE Deep Learning Container (DLC) files and quantize the DLC files to 8-bit fixed point for running on Hexagon DSP. Currently, many low precision functionalities are not available for android mobile architecture using Snapdragon processors.

In addition, Xilinx implemented INT8 deep learning operations in Field Programmable Gate Arrays (FPGA) [19]. In this paper, low precision libraries for widely used double / single precision hardware architectures have been summarized. Therefore, low precision implementation for DNN models in FPGA have been excluded in this scope.

V. CONCLUSION

In summary, the paper will contribute to a summarization of all available low precision libraries in widely used DNN frameworks for various hardware architectures. This overview will help researchers and engineers working in implementation of reduced precision numerical operations for more efficient and faster executions of DNN models in training and inference. Lower precision inference and training can improve the computational performance of DNN models with minimal or no reduction in inference accuracy. Thus, data can move faster through memory hierarchy to maximize computer resources.

REFERENCE

[1]. Micikevicius, Paulius, Sharan Narang, Jonah Alben, Gregory Frederick Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh and Hao Wu. Mixed Precision Training." CoRR abs/1710.03740, 2017.
[2]. Intel Math Kernel Library for Deep Neural networks (MKL-DNN). Available: https://intel.github.io/mkl-dnn/
[3]. Courbariaux, Matthieu et al., Training deep neural networks with low precision multiplications, 2014.
[4]. Gupta, Suyog et al., Deep Learning with Limited Numerical Precision, *ICML* 2015.
[5]. Vincent Vanhoucke and Andrew Senior and Mark Z. Mao, Improving the speed of neural networks on CPUs, Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011.
[6]. K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights +1, 0, and −1," *2014 IEEE Workshop*

on Signal Processing Systems (SiPS), Belfast, 2014, pp. 1-6.
doi: 10.1109/SiPS.2014.6986082
[7]. Miyashita, Daisuke et al., Convolutional Neural Networks using Logarithmic Data Representation, *CoRR* abs/1603.01025, 2016.
[8]. Rastegari, Mohammad et al., XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks, *ECCV*, 2016.
[9]. Mellempudi, Naveen et al., Mixed Low-precision Deep Learning Inference using Dynamic Fixed Point, *CoRR* abs/1701.08978, 2017.
[10]. Micikevicius, Paulius et al. "Mixed Precision Training." *CoRR* abs/1710.03740, 2017.
[11]. Sharan Narang, Benchmarking Deep Learning Inference, Baidu Research, 2017. Available: https://cdn.oreillystatic.com/en/assets/1/event/258/BenchmarkingdeeplearninginferencePresentation.pptx
[12]. Sze, Vivienne et al. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey." *Proceedings of the IEEE* 105, 2017.
[13]. Das, Dipankar et al. "Mixed Precision Training of Convolutional Neural Networks using Integer Operations, 2018.
[14]. Google I/O, 2018, Increasing complexity and compute needs, Stage 8. Available :https://youtu.be/vm67WcLzfvc
[15]. Nvidia TensorRT. Available : https://developer.nvidia.com/tensorrt
[16]. Intel MKL-DNN. Available : https://github.com/intel/mkl-dnn
[17]. Cortex Microcontroller Software Interface Standard (CMSIS). Available : software.github.io/CMSIS_5/General/html/index
[18]. SnapDragon Neural Processing Engine(SNPE). Available : https://developer.qualcomm.com/docs/snpe/index.html
[19]. Deep Learning with INT8 Optimization on Xilinx Devices. Available: https://www.xilinx.com/support/documentation/white_papers/wp486-deep-learning-int8.pdf

Anwarul Azim" Low Precision Arithmetic Operations in Deep Neural Networks: An Overview" The International Journal of Engineering and Science (IJES), 8.4 (2019): 39-44