Latte: Memory efficient Deep Convolutional Neural Network Framework

Ankit Tripathi
National Institute of Technology, Karnataka
13co113@nitk.edu.in

Basavaraj Talawar
National Institute of Technology, Karnataka
basavaraj@nitk.edu.in

Abstract

Latte is a light-weight framework built in C++ for inference of Deep convolutional Neural network(DCNN) that uses memory efficiently suitable for low computation and resource devices. The architecture of Latte supports non-linear networks and can be scaled easily for different network configurations. We compress the weights nearly by the factor of two at the same time our algorithm doesn't require considerable time on decompressing the weights.

1 Introduction

Computer Vision applications based on deep learning require a large amount of memory to bring the model to the DRAM and hence, latency is added even loading the weights and biases from DRAM to SRAM which requires a comparatively large amount of time when compared to the time taken for computation of that data. It has been found through experiments that 90% of the time during inference is spent in computing convolutional layers and 90% of the size of the model is used for storing weights of densely connected layers. Also, it has been observed that the output of the intermediate layers is usually higher for the convolutional layers. We are motivated to make computer vision applications based on Deep Convolutional Neural Network, real-time on low computation and memory devices.

Latte is maintained and developed by us at National Institute of Technology, Karnataka, Surathkal and we welcome open-source contributions at the GitHub link provided: https://github.com/midnitekoder/Latte.

2 Architecture

The architecture of the framework is based on the following principles-

1. Tile processing with maximum reuse of intermediate output of layers so that the architecture can handle processing irrespective of the size of the DRAM of the computing device.

2. Serial storage of the intermediate data in the buffers so that access always provide the benefits of the spatial locality.

3. The architecture can scale itself for any size of the DCNN model and also, assumes Directed Acyclic graph nature of the layers. This provides the flexibility of realizing non-linear nature of models.

2.1 Data storage

2.1.1 Weights storage

Consider a layer l of the network with k nodes. Let the dimension of weights of the nodes of the layer be $F_x \times F_y \times F_z$. Let the data that has to be multiplied and accumulated by the nodes has the same size. If the layer l is a convolutional layer then, the input is from the window it is processing. If it is fully connected layer then, the data is the output of the previous layer.

If we vectorize the weights into one dimension, we get W such that $|W| = F_x \times F_y \times F_z$. Similarly, we vectorize the input into one dimension, we get X such that $|X| = F_x \times F_y \times F_z$.

To convolve, we calculate:

$$\sum_{i=0}^{|W|-1} x_i w_i \tag{1}$$

$$w_{max} = max(|w_i|), i = 0, 1..., |W|-1 \tag{2}$$

if $\left|\frac{w_i}{w_{max}}\right| < \alpha$ given $w_i \neq 0$ Let,

$$w_i{}' = \begin{cases} w_{max} - w_i, & w_i > 0 \\ w_i + w_{max}, & w_i < 0 \end{cases}$$

therefore,

$$w_i = \begin{cases} w_{max} - w_i{}', & w_i > 0 \\ w_i{}' - w_{max}, & w_i < 0 \end{cases}$$

where α is the degree of precision in the new floating point representation which has been explained later.

$$\frac{w_i}{w_{max}} = \begin{cases} 1 - \frac{w_i{}'}{w_{max}}, & w_i > 0 \\ \frac{w_i{}'}{w_{max}} - 1, & w_i < 0 \end{cases}$$

Hence,

$$\frac{w_i}{w_{max}} = w_{ia} + w_{ib} \tag{3}$$

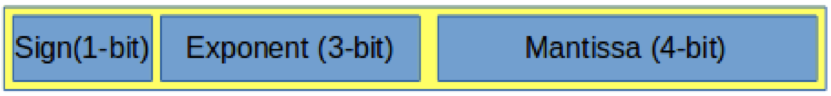

Figure 1: 8-bit floating point representation

where, w_{ia} and w_{ib} can be derived from the above equation as,

$$w_{ia} = \begin{cases} 1, & w_i > 0 \\ \frac{w_i{}'}{w_{max}}, & w_i < 0 \end{cases}$$

$$w_{ib} = \begin{cases} \frac{-w_i{}'}{w_{max}}, & w_i > 0 \\ -1, & w_i < 0 \end{cases}$$

Consider the next case,
if $\left|\frac{w_i}{w_{max}}\right| > \alpha$ given $w_i \neq 0$

$$w_{ia} = \frac{w_i}{w_{max}} \tag{4}$$

$$w_{ib} = 0 \tag{5}$$

if $w_i = 0$, then

$$w_{ia} = w_{ib} = 0 \tag{6}$$

From equation (1),

$$\sum_{i=0}^{|W|-1} x_i w_i = w_{max} \sum_{i=0}^{|W|-1} x_i \frac{w_i}{w_{max}} \tag{7}$$

$$\sum_{i=0}^{|W|-1} x_i w_i = w_{max}(\sum_{i=0}^{|W|-1} x_i w_{ia} + \sum_{i=0}^{|W|-1} x_i w_{ib}) \tag{8}$$

From eqn (1), w_{max} can be stored in Floating point single precision, but w_{ia} and w_{ib} can be stored in 8-bit format explained below.

It can be observed that w_{ia} and $w_{ib} \in [-1, 1]$. Hence, the new 8-bit format doesn't provide a provision to store numbers out of this range and hence, increases the degree of precision when compared to standard 8-bit format. Consider the Figure1, exponent stores in the range of $2^{-7} to 2^{0}$. The mantissa is 4-bit similar to standard 8-bit format.

Advantages of using this compression mechanism:

1. Achieve nearly 50 % compression and hence, time taken to bring weights from SRAM to DRAM is reduced significantly.

2. The decompression mechanism has to just type-cast weights from 8-bit to 32-bit in the DRAM for convolving with the input into two different multiplication-accumulation which can be accelerated using matrix based optimizations.

3. Range of weights of the compressed representation is same as 32-bit Single precision which would have been reduced if the weights were simply type-casted into lower size floating point representation.

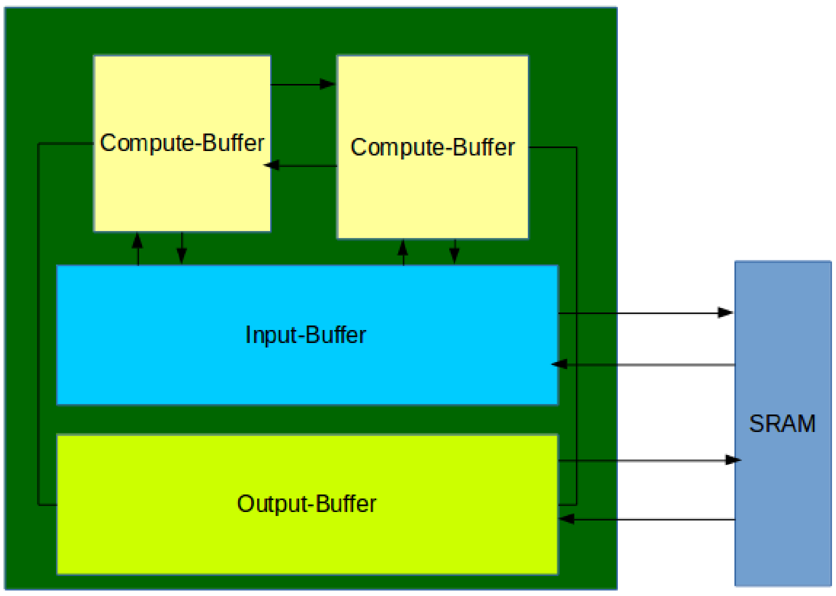

Figure 2: The architecture of Latte

2.2 Layers

Layers forward propagate the input to output. Currently, support for basic layers like Convolutional layer, Maxpooling layer, Average Pooling Layers, Local Response Normalization, Softmax layer, sparsely connected layer and densely connected layer. Optimization of computation of these layers is under progress.

2.3 Inference

Since size of the weights of fully connected layers is much larger than the size of weights of convolutional layers. Therefore, the neural network is divided into two parts by the first fully connected layer. The first part is computed using tile processing such that size of the tile is equal to the amount of input needed by the first layer of the network to produce one point of output for the first fully connected layer.

Consider the Figure2, there are two computing-buffers used for computing outputs of layers such that output of the previous layer is the input of the current layer and output of the current layer is overwritten on the output of the previous layer. Therefore, each computing- buffer stores the weights and output of a layer. The size of these buffers is decided using the intialization of the network such that it is $max(output_l + weights_l + bias_l)$ where l is the index of a layer.

There is input-buffer which is a queue such that its front always provides the next data to be processed. Whenever a layer completes producing output in the computing-buffer, the output which will be used later by the next tile is stored in the input-buffer. It is linked to the SRAM so that when this buffer overflows, the extra data can be stored in SRAM.

The output-buffer is a queue which stores the intermediate output when the further processing can be continued only after all the input to that branch is processed. And hence, it acts as a temporary storage of intermediate output. Hence, it also stores the input to the fully connected layers. In a non-linear (or branched) arrangement of layers, each branch is taken at a time and it is forward propagated till the next layer input is depended on the output of other branches. Moreover, the input-buffer stores the complete output of the node after which it is branched. Hence, the architecture can be easily scaled to any number of branches and depth.

3 Future Work

Presently, the model supports linear network and has been tested for Lenet-5 for accuracy and the results have been found to be in accordance with the results of Caffe. The different ideas presented in the paper will be experimented to provide proof-of-concept. We have a couple of algorithms to process different types of layers which would be explained in the extended paper. Currently, we are providing support to network-model trained with Caffe, though we would like to provide support for other frameworks as well. Test modules will be added and support more layers will be provided. We are focussing on the CPU based system though GPUs are yet to be explored.

References

[1] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[2] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[4] A. Lavin. Fast algorithms for convolutional neural networks. *arXiv preprint arXiv:1509.09308*, 2015.

[5] X. Liu and Y. Turakhia. Pruning of winograd and fft based convolution algorithm.

[6] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. 2011.

[7] S. Winograd. *Arithmetic complexity of computations*, volume 33. Siam, 1980.