Convolutional Neural Network Layer Reordering for Acceleration

Vijay Daultani
System Platform Labs
NEC Central Research Labs
Kawasaki, Kanagawa 211-8666
v-daultani@ax.jp.nec.com

Subhajit Chaudhury
Value Co-creation Center
NEC Central Research Labs
Kawasaki, Kanagawa 211-8666
s-chaudhury@ap.jp.nec.com

Kazuhisa Ishizaka
System Platform Labs
NEC Central Research Labs
Kawasaki, Kanagawa 211-8666
k-ishizaka@ay.jp.nec.com

Abstract— We propose a new optimization technique to speed up performance of convolutional neural networks(CNN). One of the challenges for CNN is large execution time. Many CNN models are found to have a repeatable layer pattern, i.e. convolutional, activation and pooling layer in that order. We show that, for a class of functions, performed in activation layer and pooling layer, it is possible to reconfigure CNN, to reduce the number of operations performed in a network, without changing output of the network. Experimental results demonstrate that using the proposed reordering, we can reduce the total time for VGG by almost 5% on CPU and time for activation layer, by almost 75% for CPU, and by a range of 20% to 67% for GPU, for 2x2 max pooling kernel.

1 Introduction

In computer vision many problems are solved using machine learning. Deep learning, especially CNN, in the recent past has achieved state of the art results [1] for many such problems. Due to CNN's increasing wide range of applications and its very high execution time, researchers are finding several ways to make CNN faster. Such contributions have been made both at hardware and software levels.

Hardware vendors are investing a lot of resources, in research of how to make their hardware platform to suit deep learning. Many deep learning libraries for CPU's, GPU's and FPGA's have evolved in recent past. Influence of deep learning on designing hardware has been so significant that many hardware architectures are custom built just for deep learning. TPU (Tensor Processing Unit) from Google, Fathom Neural Compute Stick from Movidius are perfect example of the same.

Many hardware accelerated libraries of primitives for deep learning are available for various hardware architectures. cuDNN [2] is one such library for GPU's. These libraries consists of highly tuned implementations of several basic routines of deep neural networks.

Although, acceleration of deep learning have been chased from both, hardware and software ends, but very few efforts, to change the configuration of deep neural networks for getting acceleration have been made. To the best of our knowledge this is the first work that shows how reconfiguration, i.e. layer reordering of activation and pooling layer of CNN model can accelerate and reduce its execution time without changing output of network.

In this paper we propose a method for obtaining speed up by reconfiguring the constituents of conventional convolutional neural networks. First in section 2 we explain the traditional convolutional neural networks and motivation for the proposed reordering of CNN model. In section 3 we present our idea of CNN model reordering. In section 4, we demonstrate mathematically how the proposed reordering can produce speed-up. In section 5 we describe the

evaluation environment for experiments and demonstrate the superiority of the proposed reordering in execution time. Finally, in section 6 we summarize and conclude the paper.

2 Background

Figure 1 shows a typical convolutional neural network configuration, i.e. how different components are stacked over each other to form a convolutional neural network.

2.1 Components

Conventional convolutional neural network is made up of the following network elements. The input layer holds the incoming image as input to pass it on to the next layers.

The convolutional layer consists of stacks of filters which are convolved with the input from the previous layer. Each of the filters produce a feature map by convolution and each feature map forms a channel of the convolutional layer output.

After producing each feature map, they are passed through a non-linearity layer(also called activation layer), which aids in the visual recognition task. Common non-linearity layers are sigmoid, tanh and Rectified linear unit(ReLU). Out of these ReLU activation has gained popularity in recent times.

To reduce the number of parameters and to provide translational invariance property to the visual recognition task, the activation maps are down-sampled by proper-down sampling ratio. Usually the down-sampling ratio is chosen to be 2, however other down-sampling ratios can also be chosen. Popular down-sampling heuristics include max-pooling, where the maximum value from a neighborhood is picked, and average pooling where the average value is picked, as the representative value in the down-sampled version.

The higher level decision about visual recognition is taken by dense layers(or fully connected layers) which are equivalent to fully connected hidden layers in artificial neural networks.

Depending upon if CNN is used to solve the classification problem or a regression problem, output of a traditional CNN model can be a soft-max non-linearity or a mean square error non-linearity respectively.

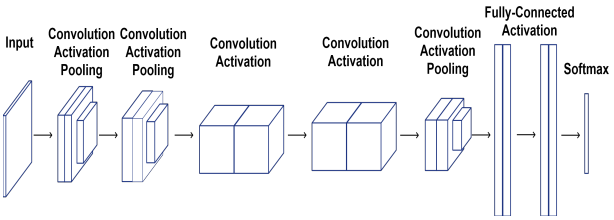

Figure 1: A typical CNN architecture showing the layers discussed in the section 2.1

2.2 Motivation

Most common form of convolutional neural networks tends to follow a pattern. These pattern can be represented by using a regular expression, like this

```
INPUT -> [[CONV -> ACT]*N -> POOL?]*M ->
[FC -> ACT]*K -> FC -> OUTPUT
```

where INPUT represents input layer, CONV represents convolutional layer, ACT represents activation layer, and FC represents fully connected layer. Moreover N, M, K are integer variables depicting the number of convolutional, total number of units and number of dense layers respectively and * in the regular expression before this variable represents that pattern before * can exists zero or more times. In the above regular expression, following part,

```
[[CONV -> ACT]*N -> POOL?]*M
```

is of special interest to us. It means that convolutional layer is always followed by activation layer, and any such occurrence may be followed by a pooling layer. Such pattern where convolutional layer followed by activation layer followed by pooling layer occurs, it can be reconfigured for acceleration, without changing output of the network. Mathematical justification for the speed up in computation is provided in section 4.

3 Proposed Reordering

We propose that interchanging the position of activation and pooling layer can reduce the number of computations performed in these layers. These reduction of computations will lead to reduction in execution time of activation layer.

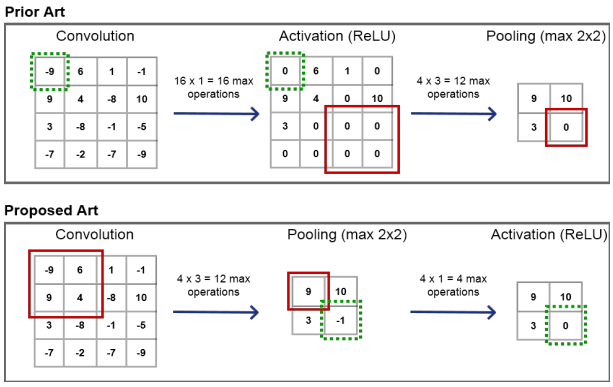

Figure 2: (a) Conventional CNN architecture showing output from convolutional layer, activation layer and pooling layer, in that order. (b) Proposed CNN architecture showing output from convolutional layer, pooling layer and activation layer, in that order.

The proposed technique is illustrated in figure 2 with ReLU(max) activation and 2×2 max pooling. As shown in the figure, we interchange the ordering of the activation and the pooling layer in the CNN architecture. In traditional setting i.e. prior art in figure 2 in activation(ReLU) layer, binary max operation is performed for each element, like max(value of element, 0) from the output of convolutional layer. Since in the shown example there are sixteen elements in the output of the convolutional layer, which leads to sixteen max operations for activation layer. Output of activation layer is then given as input to pooling layer. Since in this example we have considered 2×2 max pooling with stride of 2, there is no overlap between pooling operations. Four quaternary max operations i.e. max(0,6,9,4), max(1,0,0,10), max(3,0,0,0), and max(0,0,0,0)are performed in the pooling layer. Each quaternary max operation is broken into three binary max operations at instruction level. For example, quaternary max operation max(0,6,9,4) is broken into three binary max operations i.e. m1=max(0,6), m2=max(m1,9), m3=max(m2,4), where m3 is output of quaternary max operation. In total in prior art it leads to sixteen binary max operation for activation and twelve binary max operations for pooling, which leads to total twenty-eight binary max operations.

After reconfiguring the network, 2×2 max pooling is performed on the output of convolutional layer before activation operation in contrast to prior art. Which leads to four quaternary max operations i.e. max(-9,6,9,4), max(1,-1,-8,10), max(3,-8,-7,-2) and max(-1,-5,-7,-9). Each quaternary max operation is broken into three binary max operations leading to twelve binary max operations. Activation layer performs element wise binary max operation on output of pooling layer. Four binary max operations i.e. max(9,0), max(10,0), max(3,0), and max(-1,0). In total for proposed network configuration twelve binary max operations for max pooling and four binary max operations are performed, which leads to a total of sixteen binary max operations, in contrast to twenty-eight binary max operations in prior art.

Proposed technique of swapping activation layer and pooling layer with a condition that "**non-decreasing function for activation layer and max function for pooling layer**", was shown to reduce the number of computations in the above example. In the next section, we shown, our algorithm is mathematically correct, and does not change output of the network.

4 Mathematical Justification

Consider a particular convolutional, activation and pooling layer from the entire CNN model in its traditional setting. Without loss of generality, let us assume that $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ are the output from convolutional layer, activation layer and the pooling layer respectively. Let us represent activation layer operation by the function $f : \mathbb{R} \longmapsto \mathbb{R}$ which operates on each output value of the convolutional layer to produce the same size output. For pooling layer, let us consider $K-$pooling, such that a $K \times K$ area in the activation map will be represented by one value after

pooling. In figure 2, the value of K is 2. Representing pooling layer operation by the function $g : \mathbb{R}^{K^2} \longmapsto \mathbb{R}$ which operates on a vector of K^2 elements to produce one output(for max-pooling the maximum of K^2 values are chosen and for average-pooling, the average of K^2 values are chosen as output).

In traditional setting the final output is given as the following composition of function,

$$\boldsymbol{z} = g(f(x_1), f(x_2), ..., f(x_{k^2})) \tag{1}$$

where the inputs $x_1, x_2, ..., x_{k^2}$ represent the output of the convolutional layer in the $K \times K$ sub-grid and $f(x_1), f(x_2), ..., f(x_{k^2})$ represent the output from the activation layer.

In the proposed method, we change the order of the composition of functions and the new composition is given as,

$$\boldsymbol{z} = f(g(x_1, x_2, ..., x_{k^2})) \tag{2}$$

where first we compute the pooling operation and then perform activation on the pooled output. Next we show for which classes of functions the above assumption holds true.

We fix the pooling operation as max-pooling as it is the most popular form of pooling and other forms of pooling are very rarely used. For the max-pooling case, the function $g : \mathbb{R}^{K^2} \longmapsto \mathbb{R}$, is given as

$$g(x_1, x_2, ..., x_k) = \max(x_1, x_2, ..., x_k) \tag{3}$$

In such a case, for non decreasing activation functions, where $f(x_1) \geq f(x_2)$ if $x_1 \geq x_2$, we can successfully write the following relationship,

$$\max(f(x_1), f(x_2), ..., f(x_{k^2})) = f(\max(x_1, x_2, ..., x_{k^2})) \tag{4}$$

Consequently we found that most of the activation function used practise are non-decreasing functions for eg. tanh, sigmoid and the most popular ReLU. Thus the output of the network remains absolutely unchanged by interchanging the pooling and activation layers. Thus our proposed improvements can be used to improve time performance for practical visual recognition systems as shown in section 5.

Now we describe the improvement in timing performance by our proposed method. For $K \times K$ pooling operation, in traditional CNN architecture, activation was performed over K^2 values, whereas in the proposed architecture the activation function is performed only on 1 output value from the max-pooling operation. Thus we obtain a speed-up of $\frac{1}{K^2}$ for each activation layer unit with no change in the network output. As we increase the size of the max-pooling operation, the speed-up obtained will increase.

Layer		relu1	relu2	relu3	relu4	relu5
Theoretical speedup		4	4	4	4	4
Actual speedup VGG 16	CPU	3.9	4.0	4.0	4.0	4.0
	GPU	3.0	3.1	2.0	1.7	1.2
Actual speedup VGG 19	CPU	3.9	4.0	4.0	4.0	4.0
	GPU	3.1	2.5	1.6	1.4	1.2

Table 1: Speedup comparison for VGG 16 and VGG 19 on CPU and GPU

5 Experiments

In this section, we first describe the CNN models on which we tested our method and discuss the quantitative results for timing performance for each. We tested the proposed method on both practical visual recognition CNN models and also on our own custom built CNN architecture and show that we obtain a significant speed up in the activation layer performance on both CPU and GPU.

5.1 Implementation details

Our idea was evaluated with three different CNN models, VGG-16 layer model [3], VGG-19 layer model [3], and one customized CNN model. Customized CNN model was a simple 4 layer network, namely input, convolutional, activation (sigmoid), and a pooling(max) layer. Significance of custom CNN model is to verify that, proposed idea of non decreasing function(in this model sigmoid) for activation function is valid. Each model was inspected for pattern of convolutional, activation, pooling layer in that order and each such occurrence was replaced by convolutional, pooling, activation layer in that order. The evaluation setup consists of running 1000 iterations, in the

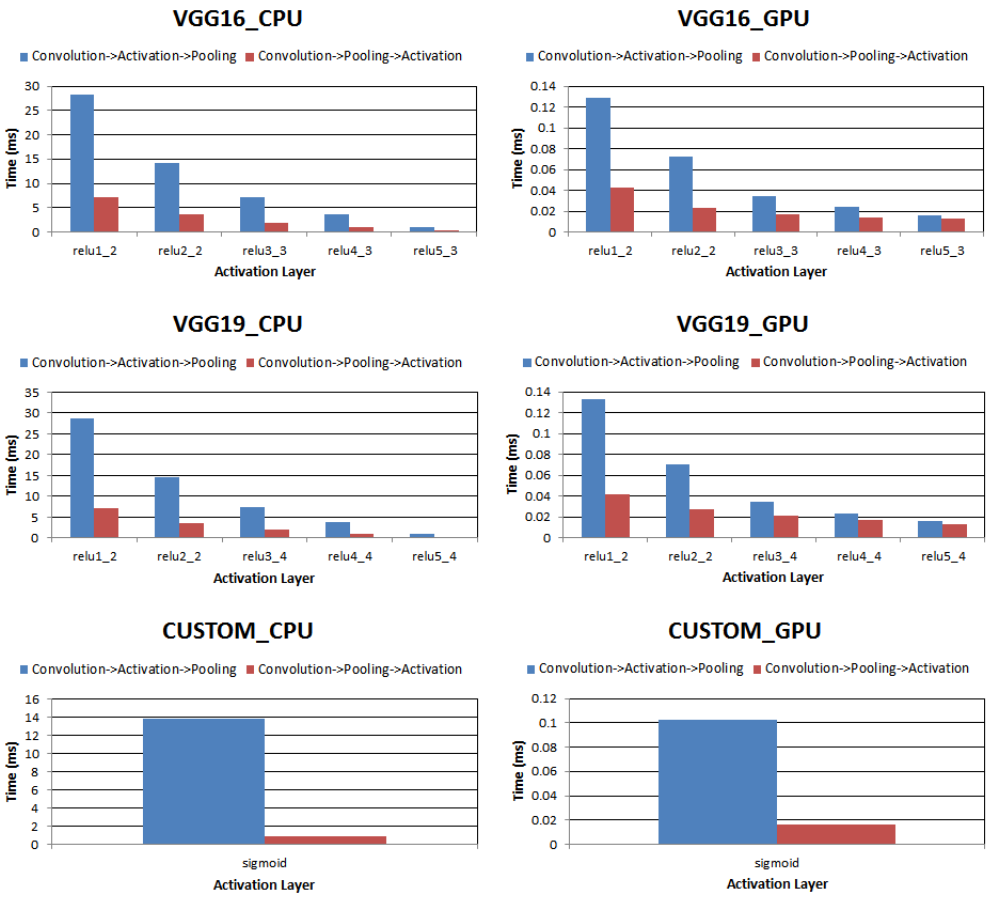

Figure 3: Time comparison graph

(a) (b)

Figure 4: (a) Prediction score for each class for Figure 4b (b) Image from [4] used for testing prediction accuracy.

testing phase of above mentioned 6 different models (VGG-16, VGG-19, customized model and 3 reconfigured models for each original model). Each model was executed for 1000 iterations, in testing phase, once on single Intel Xeon CPU E5-2660 v3 and once on single NVIDIA GTX Titan GPU. Deep learning framework used was caffe [5], with Red hat OS. Intel MKL, and NVIDIA cuDNN were used for CPU and GPU respectively.

5.2 Quantitative analysis

Since our method does not change the number of computations for convolutional and pooling layer, it was confirmed from the results that there was no change in the execution time for either of the layers. Hence in figure 3, we compare average execution time(1000 iterations) for activation layer, before and after network reordering. Figure 3 shows that time for activation layer, can be reduced almost by 75% for both VGG16 and VGG19 on CPU. This is in accordance to our mathematical justification in section 4. For GPU, the time reduction ranges from 19.56% to 67% for VGG16 and 17% to 68% for VGG19. For VGG16 and VGG19, we found that the total forward pass execution time on CPU, was reduced by almost 5% by our proposed reordering.

In future as more and more hardware architectures are going towards optimising convolution operation, the execution time for convolution operation will reduce further, and ratio of activation and pooling operations will increase further. In such a scenario our proposed reordering will have more significant total time reduction. Table 1 shows the speed up in CPU and GPU by the proposed algorithm. We investigated GPU performance with batch-size of 1, 2, 4, 8, 16 images and found that with increasing batch-size the gap between practical and theoretical speed-updecreases. This experimental results illustrate that our method is hardware independent.

Figure 4a shows accuracy predicted by VGG-16 layer on a CPU when a random image of figure 4b [4] was given as input. It can be seen in figure 4a that, both original CNN model and our reconfigured model generates exact same prediction scores for every output class. We have also verified this using 100 random images, that the output of reconfigured model matches to that of the original model.

6 Summary and Conclusions

We presented a hardware independent technique to accelerate CNN, by reconfiguring the network. Which we demonstrated by executing CNN models on both CPU and GPU. Such a technique can be easily applied to any hardware architecture. This technique is of importance because it can be applied for both training and testing phases. Since the proposed technique does not change the output of CNN model, it can be easily applied to any CNN used for non-visual inputs, given pattern of convolutional, activation, and pooling in that order exists in the network. Since many hardware architectures are being designed dedicated to deep learning, such computations reduction technique can also help to reduce the hardware logic used for deep learning.

References

[1] Alex Krizhevsky, IIya Sutskever, Geoffrey E. Hinton, " Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems 25, 2012.

[2] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, Evan Shelhamer, " cuDNN: Efficient Primitives for Deep Learning," *CoRR*, 2014

[3] Karen Simonyan, Andrew Zisserman, " Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, 2014

[4] D. Martin, C. Fowlkes, D. Tal, J. Malik, " A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics," *Proc. 8th Int'l Conf. Computer Vision*, 2001

[5] DJia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor, " Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014