Parallelized Convolutional Layer

Group 12

Wilbert
Computer Science
NCTU
0416106
wilbert.phen@gmail.com

方君安
Computer Science
NCTU
0416076
kjw472800@gmail.com

林彥傑
Computer Science
NCTU
0416213
jay101630@gmail.com

Introduction

Machine Learning has become a crucial parts of Artificial Intelligence Research. Due to the increasing amount of data and calculation power needed, parallelized parts of the layer could greatly improve overall time consumption for data training.

CNNs use a variation of multilayer perceptrons designed to require minimal pre-processing. This type of Neural Network is usually used in image classification related problems. A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.

However, there are problem, which due to the full connectivity between nodes they suffer from the curse of dimensionality, and thus do not scale well to higher resolution images.

Statements of the Problem

Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. The convolutional layer is the core building block of a CNN.

Convolutional Layer in CNN network requires matrix multiplication. Because of the high complexity of matrix multiplication in the traditional way, it take a great deal of time to compute. We hope that improving these time critical parts of the calculation could largely enhance overall performance. To do so, we can utilize the parallel computing.

Proposed Approach

In convolutional layer, primitive data has to multiply by matrix filters, in order to get the feature map. In this process, numerous number of matrix multiplication is calculated. To reduce the heavy workload, we can accelerate the process by the parallelizing the computation into several CPU cores or several CUDA cores.

When using multi-core CPU to accelerate the work, firstly, we designate the total number of work needed to be done by each core. The separation of workload is hoped to be done as fairly as possible by considering the number of cores that the current machine has and by considering the number matrix multiplication operations that need to be utilized.

Next, each CPU cores will be assigned its designated start and end requirements of its calculation to specify its portion of works and to specify the location of stored result, without relying on other parallelized parts of the code. This method is hoped to be able to decrease dependency between spawned thread and diminishing the need to use any locking mechanism to protect result data.

When using CUDA cores to accelerate the work, we first maintain the number of matrix multiplications that can be done. By calculating massive number of matrix calculation using CUDA cores, the timing is sure to be improved due to the advantage of CUDA cores in processing matrix related workload.

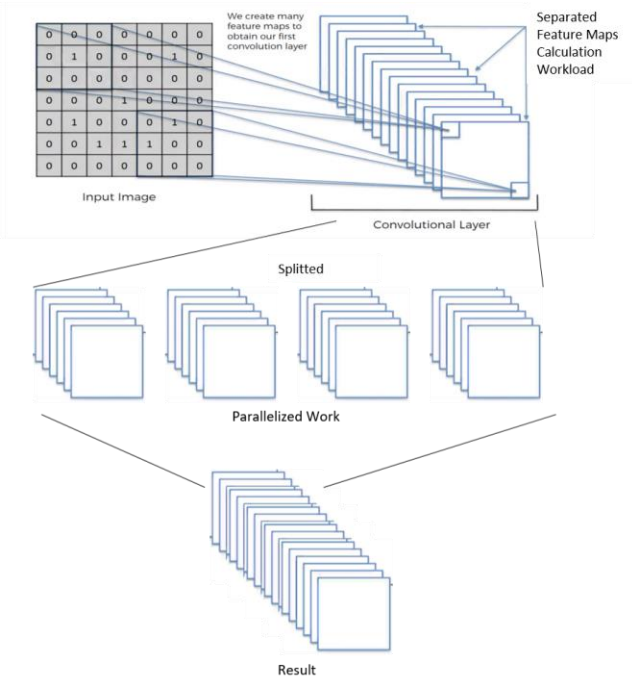

Figure 1. Simple graph visualization of the system

Language Selection

C++ or Python

Related Work

[1] Parallelizing Convolutional Neural Network Training (http://bcliu.github.io/parallel-cnn/)

Expected Result

By using CPU core, we expect the result to be better than serial version of the algorithm. CPU acceleration in this case should be able to have speedup nearly comparable to the number of CPU cores. This speedup should be achievable due to the trait of convolutional layer, which has its calculation of each resulting matrix independent from the other calculation. Therefore, result should be able to be processed in nearly perfect parallel run.

When using CUDA core, we expect the result to be better than using only CPU core with approximately 4 number of cores without any Multi-threading or Hyper-threading (Intel terms) enabled. The aim of using only pure core capabilities is to have stable comparable baseline of CPU result performance.

Timetable

1st and 2nd week	: Reading related works
3rd week	: Finished serialized parts of the code
4th and5th week	: Parallelized parts using pthread
6th and 7th week	: Parallelized parts using CUDA
8th week	: Final Report and Testing

REFERENCES

[1] Jilin Zhang, Junfeng Xiao, Jian Wan, Jianhua Yang, Yongjian Ren, Huayou Si, Li Zhou, and Hangdi Tu, A Parallel Strategy for Convolutional Neural Network Based on Heterogeneous Cluster for Mobile Information System, https://www.hindawi.com/journals/misy/2017/3824765/

[2] Dang Ha The Hien, A guide to receptive field arithmetic for Convolutional Neural Networks, https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807

[3] Keiron O'Shea and Ryan Nash, An Introduction to Convolutional Neural Networks, https://www.researchgate.net/publication/285164623_An_Introduction_to_Convolutional_Neural_Networks