Group members: Sharif Fahes, Reina Bitar, Nour Onaissi

Convolutional Neural Networks in Image Recognition Systems

# Literature Review:

Image recognition is an important technology that is being used in different fields of science and applications. Automated and self-driving cars use image recognition systems to identify objects around them. Another use is to help blind people to know what is surrounding them by using image recognition applications on their phones. Furthermore, scientists are trying to use image recognition systems in medical fields to diagnose cancer tumors earlier or even detecting covid-19 by analyzing chest and lungs x-ray images. All these applications are possible due to the use of neural networks. Neural networks are type of algorithms that works in a similar way of how human brain works in which it takes a set of data as input and then it produces an output based on some patterns. The convolutional neural network is one of the neural networks that works very well with images of large pixels and it is widely used in image recognition systems (Su, 2021). Viebke et al. (2017) explain that these networks consist of 3 layers: First, the convolutional layer where more than one filter (kernel) - represented by a matrix of numbers - convolves on the image which is represented by a matrix of pixels then it performs some computational methods resulting in a smaller matrix. Second, the pooling layer in which the previous convolved matrix is reduced again keeping the important information by using max pooling, avg pooling, or min pooling. Eventually, there is the fully connected layer, here an activation function is used - usually sigmoid function that takes a vector of real number values and reduces it to a vector that ranges between [0,1] (probability) - and based on that output a percentage of similarity is given in which the highest percentage is taken, then a backpropagation process is performed to update the weights accordingly.

Modern cameras can capture high-resolution photos which are images of very large pixels. Due to this increase in size, the time required for the training of convolutional neural networks increases. Moreover, training convolutional neural networks performs computations with a lot of floating points operations which consumes a lot of time to complete if we are using traditional single machines (Bajpai, 2015). Furthermore, Zhang et al. (2017) explain that the data sets that are being used for convolution neural networks training are very large and complex which makes the traditional algorithms inefficient to complete the training process in a reasonable time. Additionally, Su (2021) states that convolutional neural networks are very intensive regarding computational operations and can take a long execution time. Therefore, the time complexity of the convolutional neural network might be improved by parallelizing these algorithms.

With the increasing demands on implementing convolutional neural networks and the need to shorten the duration of the training, researchers had to find new ways to increase the efficiency of the algorithm and decrease its time complexity. Most of the studies have proposed to parallelize the problem through GPU implementations. For example, Bajpai (2015) had parallelized the CNN by GPU implementation using CUDA. CUDA is a parallel computing architecture in which it enables developers to use NVIDIA’s graphic processor for general-purpose GPU (Bajpai, 2015). Besides the GPU implementations, some researchers have proposed other parallel implementations. For instance, a study done by Su (2021) showed a huge increase in performance and shortened the duration of training through applying data-parallel models with some optimization for the mathematical methods that are used in the training process. Thus, different implementations have been applied to parallelize the problem and these implementations have shown promising results for speeding up the training process of convnets.

In conclusion, convolutional neural network is type of neural networks that works efficiently on images with very large pixels. It is widely used in image recognition systems which are used in numerous fields such as self-driving cars and medical fields. This type of networks requires heavy computation operations and takes a long time to be trained if we are using serial implementations. From there, many researchers proposed different implementations to parallelize the problem.

A picture containing text, crossword puzzle

Description automatically generated

Fig. [1] Convolution in serial implementation

Src: rubikscode

# Our Approach:

In our project, we attempted to parallelize the first 2 layers of CNN which is called the feature learning (convolution and pooling layer) [Fig. 2] through MPI and pthreads. Our propose is to calculate the number of rows that will be produced after convolution through a mathematical formula [Fig. 3]. Then, we divide these rows equally among each process in which each process will be computing the elements that form these rows. Considering that the image may need to be convolved by more than one filter, then whenever the process finishes its convolving of the first filter it moves to the next one without the need to wait for the other processes to finish (their computations are independent). Eventually, when all processes finish their convolving, their output will be passed into the max pooling layer. In the next layer (max pooling), we will calculate the number of rows that will be produced after the max pooling by the same mathematical formula used before [Fig. 3]. Then, for each row we will create a thread and performs a max pooling operation on it.

[Fig. 4 clarifies the convolutional process on 1 channel and 1 filter]

Diagram, engineering drawing

Description automatically generated

Fig. [2] Convnets layers

Text

Description automatically generated with medium confidence

nin: image dimension / f: filter dimension / nout: result dimension / s: stride

Fig. [3]

Calendar

Description automatically generated with low confidence Fig. [4] parallelized convolution.

# Testing:

Our experiments are tested on 2.3 GHz 8-Core Intel Core i9 device. In order to measure our algorithm efficiency, we used different combinations of input and measured the speedup. In the input, we varied the dimension of filters, the dimension of image, and the number of filters. We varied one variable at a time so when changing one variable we kept the others fixed. Eventually, with each new input we tested it on 1, 2, 4, and 8 processors and measured the execution time.

# Result:

1. Varying the size of filter:

* 1 channel (grey image)

Chart, bar chart

Description automatically generated

Fig. A1

* 3 channels (colored channel):

Chart, bar chart

Description automatically generated

Fig. A2

1. Varying the size of image:

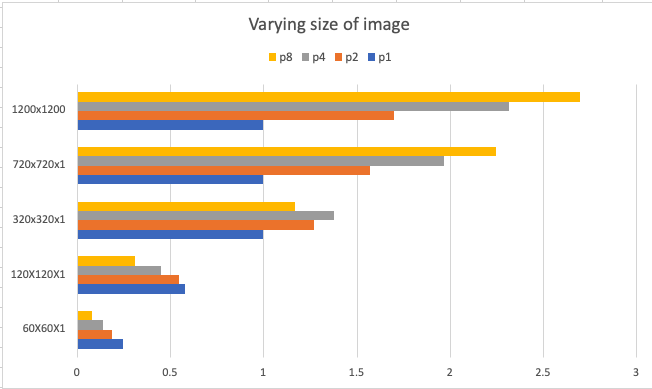


Fig. B

1. Varying the number of filters:

Chart, bar chart

Description automatically generated

Fig. C

# Conclusion:

In figure A1 and A2, we are testing the variety of filter size on 800 x 800 x 1 image size. We can notice the small speedup values on small filter size such as 2x2x1 filter. This may be due to the fact that with smaller size of kernel more processors are being idle which decrease the efficiency of parallelization. On larger input values of kernel, we can notice the increase of speedup with the increase of number of processors. Thus, the algorithm is efficient with large size of filters.

In figure B, we are testing the variety of image size using one 10x10x1 filter. We can notice the same results as the previous testing. The speedup is increasing with the increase of number of processors starting from image size 320x320x1. Therefore, the efficiency the algorithm is on large image size.

In figure C, we are testing the variety of number of filters with dimension 10x10x1 on 800x800x1 image. We can notice with the increase of number of filters the speedup is increasing and the parallelizing is being more efficient.

In conclusion, we can see from the previous testing results that our algorithm is efficient on large sizes and can save a lot of time than using serial implementation for convolving and pooling layers.

# Future Improvements:

* Our algorithm can be modified to permit multiple iteration for the convolutional layer.
* Convolving on channels can be parallelized using threads in which each thread convolves around 1 channel then their results are combined
* Our algorithm can be combined with the classification layer (fully connected layer) for final results.

**References:**

Bajpai S. (2015). Implementing Convolutional Neural Network with Parallel Computing using CUDA. *International Journal of Innovative Science, Engineering & Technology,* 2(11), 517-519. www.ijiset.com

Su, Y. (2021). A parallel computing and mathematical method optimization of CNN network convolution. *Microprocessors and Microsystems,* 80(103571), 2-4. https://doi.org/10.1016/j.micpro.2020.103571

Viebke, A., Memeti, S., & Pllana, S. (2017). CHAOS: a parallelization scheme for training convolutional neural networks on Intel Xeon Phi. *Springer,* 1-31. DOI 10.1007/s11227-017-1994-x

Zhang, J., Xiao, J., Wan, J., Yang, J., Ren, Y., Si, H., Zhou, Li., & Tu, H. (2017). A Parallel Strategy for Convolutional Neural Network Based on Heterogeneous Cluster for Mobile Information System. *Hindawi,* 1-12. https://doi.org/10.1155/2017/3824765