MapReduce-based Deep Learning With Handwritten Digit Recognition Case Study

Nada Basit Yutong Zhang Hao Wu Haoran Liu
Jieming Bin Abdeltawab Hendawi Yijun He

University of Virginia, VA, USA
{basit, yz5rv, hw6ca, hl4fb, jb5ej, hendawi, yh9vm}@virginia.edu

***Abstract*—Faced with continuously increasing scale of data and expectation on response time, complex deep learning technologies, though proud of high accuracy, present two non-rival challenges: huge amount of training data makes a model impossible to be built in short time and intolerable time-cost prohibits acceptable real-time responses. In this research we are focusing on the improvement of the accuracy and efficiency of the handwritten digit recognition problem. We chose this problem because it is regarded as the prototype of many complex recognition and classification problems. The success of classification of the handwritten digit dataset can be extended further to other advanced areas. The Convolutional Neural Network (CNN) is implemented to do the recognition. We further improved the accuracy by adding elastic distortion to the input data, which helps the model better select the features. Secondly we implement distributed computing to reduce the time cost. The training process is divided and a final model is formulated by the combination of each trained model. The results shows two facts: the elastic distortion helped the CNN model to improve the accuracy by 7-10%; and the distributed computing method reduced the training time consumption by about 50%.**

I. Introduction

Handwriting recognition is the ability of a computer to retrieve and interpret handwriting input from sources such as paper documents, photographs, touch-screens and other devices. It is a classic machine learning problem, and the ideas have been applied to computer vision, speech, natural language processing, and other domains [5], [6]. For example, digit recognition has been employed regularly by the post office since the 1960s for the purposes of classifying mailing addresses using Optical Character Recognition (OCR) [1].

In this paper, we investigate the digit recognition problem using a deep learning approach that is implemented using a distributed computing model to improve the efficiency.

A widely used and recognized method to solve such problems is the neural network (NN). We decided to implement a modern deep learning model, choosing to use a convolutional neural network (CNN). In machine learning, a CNN is a type of feed-forward artificial neural network, whose individual neurons are arranged in a unique way such that they respond to overlapping regions tiling the visual field. CNNs, inspired by biological processes, are variations of multilayer perceptrons (MLPs) that consist of multiple layers of small neuron collections which process portions of the input image. Memory reduction and improved classification performance can be achieved by using CNNs because of the shared weight in the convolutional layers, allowing the same filter to be used for each pixel in the layer [12]. Compared to other image classification algorithms, CNNs use relatively little pre-processing. The network itself is responsible for learning the previously hand-engineered filters. The lack of dependence on prior knowledge and human effort in designing features is another major advantage for CNNs. Due to its many advantages, it is not surprising that CNNs, as an advanced deep learning technology, have wide applications in many domain areas.

However, besides good performance, convolutional computation is generally very costly in terms of time, which is intolerable in some conditions. For example, if we add more than one convolution layer and set plenty of neurons for each fully-connected layer, the training process then becomes pretty time-consuming and with limited computation resources, it is not that trivial to solve this problem anymore. Algorithms in general are expected to find a balance between performance and time cost. A fast but inaccurate performance is equally undesirable as a slow but accurate performance. CNN, along with many other well known algorithms, such as Deep Neural Network (DNN) and Recurrent Neural Network (RNN), are forced to deal with time performance optimization. For CNN in particular, this dilemma is made worse not only from CNN's construction, but also from the large scale of input data used. More training data is sure to provide the network with more information however this also increases the chance to exhaust resources.

Much work focusing on handwritten digit recognition with large datasets have used neural networks [20] due to the expensive computation feature of CNN. Neural networks are expected to generate satisfying results on normal dataset. However, with elastic distortion, one kind of noise common in handwritten image, CNN is necessary to preserve accuracy. So far much research has been done in order to improve the efficiency. In one recent paper [20], they have tried to implement the Neural Network by MapReduce. Other research has been done using a parallel computing method, such as GPU-based computation, and multi-processes computation.

Our proposed solution focuses on tackling the time cost

limitation of CNN by using distributed computing; the data is distributed across machines in a Map phase, then weights are combined to formulate the final model in the Reduce phase. MapReduce is a straightforward programming framework tool for implementation of distribution computation.

First, we succeed in implementing the CNN model with elastic distortion. The distortion helps the training process to better capture the feature of the image dataset, which results in a higher accuracy in total. In practice, the elastic distortion is very common in the handwritten digit images, so our implementation here builds a helpful application. Second, the most improvement of our research is to use distributed computing method to do the training process in CNN, which gives a more efficient way to use the application. Based on our experiment, our model reduces the computation time by about 50%, and the efficiency varies by different number of machine processes we use.

Now, we talk about why the distributed training processes would result in one valid model. In CNN, the gradient descent method is used to update the parameters each time. In other words, with each training iteration the direction that mostly optimizes the training data is calculated and the new parameters are moved towards that direction. Thus the direction from the initial parameters to the final one is equivalent to a combination of a series of vectors. With distributed computing, we decompose the vectors at the initial point, and they are combined together after the distributed computing is finished, and the final weights are not change with or without the distribution.

The first kind of parameter is the number of layers in the neural networks, we try different values to see how it influences the efficiency on distributed computing. The second kind of parameter is how many distributed machines we use, since this also affects the efficiency due to the data communication between machines.

In this research, we propose a method to improve the efficiency of CNN model based on distributed computation. Towards that end we carried out empirical analysis on the effects of changing different computing machines and different layers in neural networks. Our result shows that this method can indeed reduce the time cost while the accuracy remains at a reasonable level.

In the following sections, we first review typical classification algorithms which have been previously used to classify handwritten digit numbers and how distributed computing helps reduce execution time (Section II). After that, we clarify the problems we are focusing on and present the outline of our solutions (Section III). Then, we explain our proposed solution, the CNN classification algorithm with distributed computing, in detail (Section IV). After that, our experimental design and comparative results are presented (Section V). Finally, we conclude the paper in Section VI.

II. Related Work

Stephen J. Smith and his team [19] built a simple statistical technique and a large training database to automatically optimize to produce high classification accuracy in the domain of handwritten digits. Three distance metrics for the standard Nearest Neighbor classification system are investigated: a simple Hamming distance metric, a pixel distance metric, and a metric based on the extraction of features.

The authors in [21] present a hybrid KNN-SVM method for cursive character recognition. Specialized Support Vector Machines (SVMs) are introduced to significantly improve the performance of KNN in handwritten recognition. This hybrid approach is based on the observation that when using KNN in the task of handwritten characters recognition, the correct class is almost always one of the two nearest neighbors of the KNN. Specialized local SVMs are introduced to detect the correct class among these two different classification hypotheses. The hybrid KNN-SVM recognizer showed significant improvement in terms of recognition rate compared with MLP, KNN and a hybrid MLP-SVM approach for a task of character recognition.

XiaoXiao Niu and Ching Y. Suen [17] designed a hybrid CNN SVM model for handwritten digit recognition [8]. This model automatically retrieves features based on the CNN architecture, and recognizes the unknown pattern using the SVM recognizer. High reliabilities of the proposed systems have been achieved by a rejection rule.

On Handwritten digit recognition by neural networks with single-layer training [9], S. Knerr and team demonstrate that neural network classifiers with single layer training can be used efficiently to complex real-world classification problems such as the recognition of handwritten digits. They introduce the STEPNET procedure, which decomposes the problem into simpler sub-problems which can be solved by linear separators. They present results from two different data bases: a European database comprising 8700 isolated digits, and a zip code database from the U.S. Postal Service comprising 9000 segmented digits. Finally, they show that their network, resulting from the STEPNET building and training procedure and using an appropriate data representation, leads to very satisfactory recognition rates on two moderately sized data bases of handwritten digits.

Y. LeCun and his team, on Backpropagation Applied to Handwritten Zip Code Recognition [11], states that the ability of learning networks to generalize can be greatly enhanced by providing constraints from the task domain. They demonstrate how such constraints can be integrated into a backpropagation network through the architecture of the network.

Most of the work reviewed did not introduce elastic distortion to the original data set. We decided to integrate this idea into our research and found that our final results were improved (in therms of accuracy.) Furthermore, most of the work reviewed on handwritten digit recognition did not use the MapReduce paradigm. We successfully used Hadoop to improve our training efficiency without losing accuracy.

III. Preliminaries

A. Background

Handwriting recognition is the ability of a computer to retrieve and interpret handwriting input from sources such

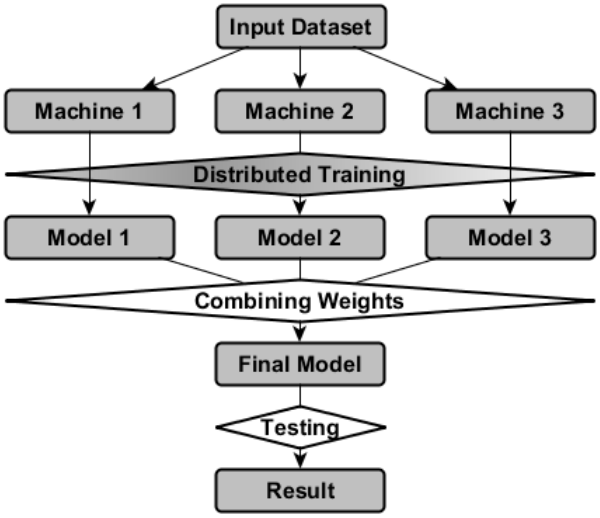

Fig. 1. Distributed Computing Architecture

as paper documents, photographs, touch-screens and other devices. It is a classic machine learning problem [4][16] with roots at least as far as the early 1900s, and the ideas have been applied to computer vision, speech, natural language processing, and other domains. For example, digit recognition has been employed regularly by the post office of all organizations, since the 1960s for the purposes of classifying street addresses using Optical Character Recognition.

Many algorithms have been proposed to solve this problem. Theoretically, every machine learning classification approach can be applied to this problem. From the basic linear classifier to advanced SVM approach, to recent neutral network approach, people keep improving the performance of handwritten digit classifier [18]. One reason why people put in so much passion in this problem is because handwritten digit recognition can be regarded as the prototype of many complex recognition and classification problems. The success of classification of the handwritten digit dataset can be extended further to other advanced areas.

Recently, deep learning [7][2] theory has been proposed to solve traditional machine learning problems, including image classification. One of the most important algorithms in this category is convolutional neutral network (CNN). It combines convolution operation with neutral network to extract local feature of 2D signal such as images and audio, and achieves quite promising result. In just a few years, such approach has been applied into many areas to solve different kind of problem.

There is no doubt about the performance of CNN, the problem is in actual application, it is intolerable to take one hour to recognize a ten-thousand article. On the condition of the high accuracy, time-cost is a significant concern before this algorithm goes out of labs. A balance between accuracy and cost is usual dilemma for classifier. To CNN its even more serious due to its incredible construction and calculation complexity.

Distributed computing system [3][10] is common solution for big data. The idea is that we use multi-process on different CPUs with different machines [15], which reduces the computations time in total [14]. With proper implementation, we could separate the dataset and then combine the computed result together to make a reasonable model. Therefore, we decide to use this distributed computing to implement our CNN algorithm.

Fig. 1 shows our proposed solution. We begin by discretizing the input dataset and map each batch onto different machines. By CPU-based parallel computation [13], we then train the CNN model with each batch of training data. After the training process, the parameters of the individual models are combined, creating one single (final) model. Finally, we test the final model to determine the accuracy and analyze the efficiency.

B. Problem Definition

1) Inputs: The input of our model is the handwritten digit dataset from the most widely used dataset website MNIST. It could be obtained from *http://yann.lecun.com/exdb/mnist/*.

The original black and white images from MNIST were normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the antialiasing technique used by the normalization algorithm. These images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

2) Outputs: Given the input image, The output would be the classification result from the CNN model. To be more specific, the output is an integer from 1 to 9 inclusively, which indicates the digit in the image recognized by the algorithm.

In addition, in order to see the accuracy and efficiency, we also output the training time for certain number of iterations, and the accuracy after the entire training and testing process.

3) Research Question: With respect to improving classification problems implemented using a convolutional neural network (CNN), our research is centered around two important questions: (1) Can the classification accuracy be improved by introducing elastic distortion and noise to the original dataset? and (2) Does combining a distributed computing technique with the CNN model improve classification efficiency?

The handwritten digit recognition problem is the chosen case study upon which these questions are explored.

4) Objectives: The objective is mainly about the accuracy and efficiency. Based on the elastic distortion, we try to improve the feature selection process, which makes the model more robust and improves the accuracy. As for the efficiency, the distribution would help us to reduce the computation time, so in future work more complicated model could be constructed if the computation complexity could be solved here.

IV. Proposed Solution

In this section we discuss our proposed solution along with various aspects that contribute to the solution. First the

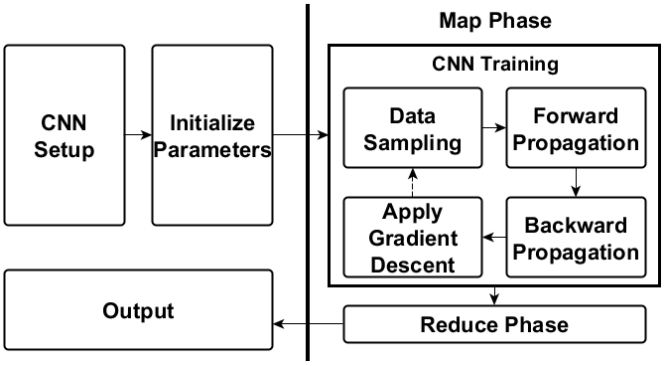

Fig. 2. Solution Architecture for CNN with Distributed Computing

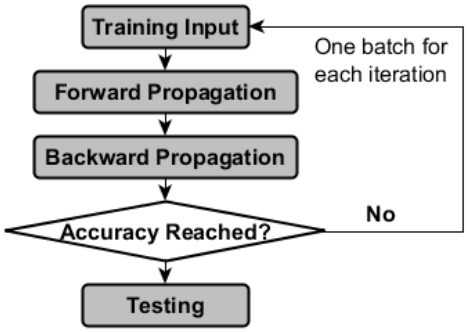

Fig. 3. Naive Training Framework

solution architecture is discussed followed by our main idea (our proposed solution). Next our algorithm is presented. In the next two sections we explain convolutional layers and pooling as well as backpropagation which play central roles in our solution and algorithm. We finally present a brief analysis on cost and correctness.

A. Solution Architecture

First, we implement the naive CNN algorithm with single process, as it is originally used. The CNN part is implemented by Python with Tensorflow library. Tensorflow is an open source Python library developed by Google, which helps users to implement the deep learning algorithms in a more efficient way. CNN training consists of data sampling followed by forward propagation and then back propagation. Weights are adjusted via gradient descent. Second, we add the distributed computing to the naive model. This process is also implemented by Python. The reason why we try to use the same programming language is to facilitate the calling and communication process between different languages which in turn will not require extra time. Using different programming languages may require extra computation time, which is a disadvantage. With the distributed computing model, CNN training is distributed across machines in the Map Phase; the weights are then combined to formulate the final model in the Reduce Phase. The final output (result) is then produced.

B. Main Idea

The naive CNN model (see Fig. 3 is a simple non-distributed model which acts as a baseline that we use to compare our proposed distributed computing model. Based on the naive CNN model, we can already reach a reasonable accuracy. However, the training process costs much time and in this module we want to reduce the time consumption by distributed computing. Thus we implemented this method to map different parts of the input dataset into different machines and then do the training process simultaneously. After that we collect all the weight matrices and combine them to formulate a final model. (See Fig. 4)

C. Algorithm

Unlike the baseline naive CNN model mentioned in the previous section, in this strategy, distributed computing is injected directly into the convolutional neural network. Due to the highly complex structure of the CNN model, a distributed computing method is implemented to handle the large amount of computation involved.

In the distribution block, each machine would be assigned to one batch of input, and they will simultaneously execute the training process individually, which has already been implemented in the first module. In other words, each machine also goes through the convolutional process, pooling process, and the fully-connected process. After that the back propagation is used to update the parameters individually based on the same initial weights, which is explained by Fig. 3.

In the combination block, we collect all the trained neural network model, and combine their parameters together with vector operations. To be more specific, the matrices used to store the weights are collected to one machine and then the corresponding values are added based on the matrix operation. Then we obtain the final model.

Finally, we do the testing process with the testing dataset to see if they are correctly combined and analyze the time consumption in the experiment section.

D. Convolutional Layers and Pooling

CNN is a special case of neural network. Compared to normal neutral network it has two specialities. The first is some layers are not fully connected to its previous or next layer. The second is some clusters of nodes share common weights. These two specialities are generated because two special layers exist in CNN, which are called convolutional layer and pooling layer.

Convolution is an important operation for image processing. Applying convolution with a convolutional kernel is essentially a filter process. The basic math presentation for convolution is:

$$f(x,y) * g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \quad (1)$$

$I = f(x,y)$ is an input image, $f(x,y)$ is the gray value for the pixel at row x and column y. $w(x,y)$ is called convolutional

Algorithm 1 Distributed Computing CNN Training Algorithm

```
Get path()
Input ← image
Initialize neural network weights
for each input batch :
        Assign it to one machine
for each machine :
        Get initial weights
        Training Process
        for each training iteration :
                Read One Batch Input
                Start training process :
                        Forward propapagtion :
                        Convolution process
                        Sampling process
                        Fully connected process
                Backward propapagtion :
                        Calculate gradient descent
                        Weight update
                offset ← backward propagation()
Collect all weight matrices to main machine
Combine the corresponding weights
Final parameters ← combined parameters
Testing Process
for each testing iteration :
        do read one image
execute CNN model
output ← recognized digit
```

kernel, while a and b define the size of $w(x, y)$.

We can see from Eq (1) that convolution is essentially a weighted summary operation. The kernel slides on the image, aligns its center to each pixel and calculates the weighted average for covered region as the response of this pixel.

The reason why such convolutional layers are introduced into neutral network is that, human being's visual recognizing process is hierarchical. Lower layers extract some edge features while higher layers capture shape and motion information. From low layer to high layer, extracted features become more and more abstract. Since in computer vision, image feature extraction relies on convolution, we add two or three convolutional layers to simulate this procedure and extract different kinds of features from input images. Each convolutional layer is an image convolution of the previous layer, where the weights specify the convolution filter.

Another advantage of convolutional layer is weight-sharing. Suppose the size of input image is 13×13, which means the input layer has 169 neutrons. For fully connected layer, with only 20 neutrons, we have to set $(169+1) \times 20$ weights. Convolutional layers reduce the number of weights dramatically. If we use 5×5 kernel, and generate 6 feature maps for this convolutional layer, we only have to set $(5 \times 5+1) \times 6$ weights.

If we denote the k-th feature map at a given layer as h^k, whose filters are determined by the weights W^k and bias b_k,

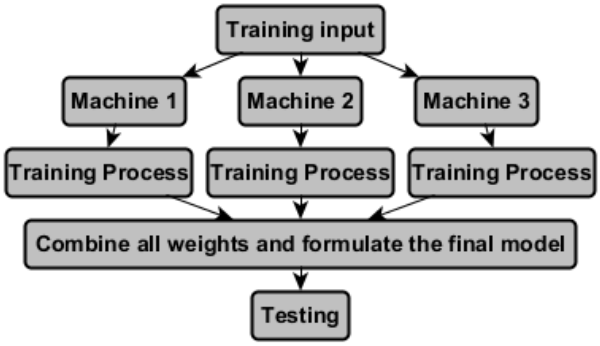

Fig. 4. Distributed Training Framework

then the feature map h^k is obtained as follows (F denotes a non-linear active function):

$$h_{ij}^k = F((W^k * x)_{ij} + b_k) \tag{2}$$

$k \in [0, k]$ means we have k feature maps as output as well as k kernels for convolution.

After each convolutional layer, there may be a pooling layer. This is because using all extracted features from the previous convolutional layer would be quite challenging. The aim is to reduce such a large number of neutrons that would increase computation complexity to unacceptable levels. Reducing the number of features would also reduce the possibility of over-fitting.

To solve such problems, statistics from the feature maps are collected, and make each feature represent a local region instead of a single pixel. This process is called down-pooling. Generally, there are two main strategies for pooling that are used: max-pooling and mean-pooling. Both of them first partition the output feature maps of previous convolutional layer (i.e. the input into the current pooling layer) into non-overlapping rectangles. For each sub-region, max-pooling takes the maximum value as the output and mean-pooling takes the average value as the output. By doing this, a single output is produced for that block which reduces the difficulty of the problem.

E. Back propagation for CNN

Back propagation (BP) for fully connected layers, such as in a neural network, is straightforward. However, in this algorithm, with the presence of convolutional and pooling layers, the situation is slightly different [5]. Let δ^{l+1} be the error term for the $(l+1)$-st layer in the network with a cost function $J(W, b; x, y)$ where (W, b) are the parameters and (x, y) are the training data and label pairs. If the l-th layer is a pooling layer then the error is propagated through as:

$$\delta_k^{(l)} = upsample(\delta_k^{(l+1)}) \tag{3}$$

Eq (3) illustrates the process of reverse max pooling. The upsample operation propagates the error through the pooling layer by putting the error from layer $l+1$ to layer l. During forward pooling, we record which pixel is the local maximum.

During reverse pooling, we put the error back to that local maximum position and set all the other positions as 0.

If the l-th layer is a convolutional layer then the error is propagated through as:

$$\delta_k^{(l)} = ((W_{(k)}^l)^{\mathrm{T}} \delta_k^{(l+1)}) \cdot f'(z_k^l) \tag{4}$$

The main difference from BP in fully connected layer is, we need to choose the correct weight matrix according to which convolution kernel generates the node with $\delta_k^{(l+1)}$ error.

Once we get the error term for each node in the CNN, we can take a step further to calculate the gradient w.r.t to the weights and bias. we rely on the border handling convolution operation again and flip the error matrix δ_k^l the same way we flip the filters in the convolutional layer.

$$\begin{aligned} \nabla_{W_k^{(l)}} J(W,b;x,y) = \sum_{i=1}^{m} (a_i^{(l)}) * \\ rot90(\delta_k^{(l+1)}, 2), \nabla_{b_k^{(l)}} J(W,b;x,y) = \\ \sum_{i=1}^{a,b} (\delta_k^{(l+1)})_{a,b}. \end{aligned} \tag{5}$$

Where $a^{(l)}$ is the input to the l-th layer, and $a^{(1)}$ is the input image. The operation $(a_i^{(l)}) * \delta(l+1)_k$ is the "valid" convolution between i-th input in the l-th layer and the error w.r.t. the k-th filter.

F. Expanding Data Sets through Elastic Distortions

Given a classification task, one may apply transformations to generate additional data and let the learning algorithm infer the transformation invariance. [13] states that by introducing elastic distortion to original data set, they get better result than approach with simple affine transformation. Following their strategy, we integrate this part into our project, to justify its validation and improve our final result.

For every pixel in the original image, we compute a new target location with respect to the original location. The new target location, at position (x, y) is given with respect to the previous position. For instance, if $D_{(x,y)}^x = 1$ and $D_{(x,y)}^y = 0.5$, it means the new location of (x, y) is shifted by 1 to the right and 0.5 to the bottom. If the displacement field is $D_{(x,y)}^x = \alpha x$ and $D_{(x,y)}^y = \alpha y$, that means the displacement is scaled by α from the origin location.

As initialization, we randomly generate values for vertical and horizontal displacement. That is, we first set $D_{(x,y)}^x = rand(-1, 1)$ and $D_{(x,y)}^y = rand(-1, 1)$, where $rand(-1, 1)$ us a random number between -1 and +1, generated with a uniform distribution. The fields D^x and D^y are then convolved with a Gaussian filter of a standard deviation σ. The displacement of every pixel forms two displacement maps (vertical and horizontal). They are then scaled by a scaling factor α that controls the intensity of the distortion.

(a) (b)

Fig. 5. Two original hand written digit images, the number 6 (a) and the number 4 (b), each with two displacement maps (vertical and horizontal).

G. Cost And Correctness Analysis

Now we briefly describe the cost of the model. Originally, we assumed the training process of the naive CNN costs the time of n and the testing process costs the time of m. Then in our model, we try to reduce the time consumption by distributed computing. If we use two process parallel computing, which means we assigns the input dataset into two different machines and do the training processes simultaneously, then theoretically the time cost of the training process in our model would be $n/2$. Moreover, if we assign the input dataset into p different machines, then the training process costs n/p. Thus, in theory, the total time cost would be:

$$n/p + m \tag{6}$$

However, we also need to consider the data communication between different machines in our project. Since in the combination step, we collect all the parameters in each model and calculate the final parameters, this part of time consumption is necessary. In our result we could see that the time cost of our model cannot reach the ideal situation, which is half of the original time consumption.

V. Experimental Evaluation

In the empirical analysis, we evaluate the performance, accuracy and efficiency of the Convolutional Neural Network algorithm. In the following sections, we will discuss the experimental setup, how we did the experiments and the corresponding results.

Based on the two models discussed above, we implement those in the experiments to see their performance. On the first step, we implement the naive CNN model by using the Tensorflow library. In this step we set all the necessary parameters explicitly so that we could do several evaluations later. On the second step, we distribute the training input into several machines, and do the training process individually. Then all the weight matrices are sent to the main machine so that the combined parameters could be calculated. Finally, we use the final model to test the accuracy, which indicates the correctness of our combination.

In these experiments, we are mainly concerned with accuracy of the classification and the efficiency of the algorithm. Since we use the same training and testing dataset, these two methods indicates the same accuracy if we implement the distributed computing correctly. Moreover, we analyze the time consumption by the training process, which is the main improvement we made in this project. If the time cost has been reduced with the distributed computing technique, then our project is successful.

A. Competitive Approach

In our project, we implement two different CNN training method, the single process training and multi-process training. We compare these two methods to see if the combination of the weights in the distributed computing is correctly done. More importantly, we would record the time consumption and compare these two methods mainly by the time consumption.

B. Experimental Setup

We used the most widely used handwritten digit dataset MNIST. The MNIST database has a training set of 60,000 examples, including train images and train labels, as well as a test set of 10,000 examples including images and labels. The database is a subset of a larger set called MNIST. Rather than using the original images, all digits have been size-normalized and centered in a fixed-size image. The MNIST database is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

The workload can be divided into two parts. First we implement the Convolutional Neural Network by hand. This includes the construction of the convolution layer, pooling layer, fully-connected layer, and the output layer.

By training the CNN, we estimate the parameters, which are the weights in each neuron. The workload is pretty large since we need to iteratively optimize the parameters using stochastic gradient descent. To be more specific, during the back propagation, we calculate the error term for each parameter and then update each one backwards.

For the second part, we firstly connect all the terminals together and then try if they could communicate with each other correctly. Then we implement distribution method based on the framework in the first step.

As for the parameters, we change the number of convolution layers in the CNN model in order to see the different performance: the accuracy and the efficiency. Theoretically, with more convolution layers, the model becomes more accuracy since it could better select the features in the images, while it also takes a much longer time to train the model.

The other factors that influences our experiment is the number of terminals that we use for distributed computing. Firstly we test our distribution computing method in single machine, and if it works, we then apply it with 2, 4 machines to see how it performs.

For the Convolutional Neural Network, we use Python to implement this model. To be more specific, we use the Tensorflow library to better construct the CNN model. For the distributed computing we also use Python to write the framework to do the distribution.

We have 4 terminals in total that were used in the experiment. We don't use them all in every experiment, and the number varies based on the requirement. The description of operating system and hardware is as follows:

Terminal 1 (main): Windows 8.1, 64-bit operating system based on 64x CPU; Intel Core i5-4210U, 1.70 GHz CPU; and 6.00 GB RAM. Terminal 2: Windows 8.1, 64-bit operating system based on 64x CPU; Intel Core i5-4405U, 2.10 GHz CPU; and 6.00 GB RAM. Terminal 3: Windows 8.1, 64-bit operating system based on 64x CPU; Intel Core i7-6700HQ, 3.50 GHz CPU; and 8.00 GB RAM. Terminal 4: Windows 8.1, 64-bit operating system based on 64x CPU; Intel Core i5-6200U, 2.80 GHz CPU; and 6.00 GB RAM.

		Elastic Training Set	Normal Training Set
Elastic Testing Set	1 conv layer	25.76%	34.67%
	2 conv layers	20.12%	27.02%
Normal Testing Set	1 conv layer	6.66%	6.86%
	2 conv layers	2.53%	4.89%

Fig. 6. Case study error rate based on different layers and different input data.

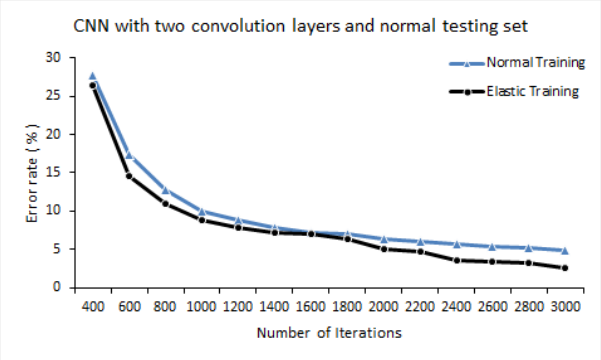

Fig. 7. Convergence for Normal Training VS Elastic Training.

C. Experiment 1

In the first experiment, we use naive CNN model to verify how the neural network performs when dealing with the handwritten digit recognition with or without the elastic distortion. In order to make reasonable comparison, we used both normal and elastic data set, and also different number of layers in the model. Fig. 6 shows the accuracy results in different cases:

Case 1: Normal Training vs Elastic Training: First we used the CNN with one convolution layers and use normal testing data set. In this case we want to see if the elastic distortion would improve the accuracy, thus we compared the elastic training data and normal training data. From the third row of Fig. 6, we could see that the elastic training set could slightly improve the accuracy.

Now we want to use two convolution layers. In the last row of Fig. 6, we can see two facts: 1) the one more convolution layer can improve accuracy in both training data sets; 2) The influence from the elastic distortion is bigger when there are two convolution layers than one.

We also want to know the convergence rate when using normal training data set and elastic training data set. As we can see from Fig. 7, these two training data sets mainly gives the same convergence rate, and the accuracy is stable after about 2000 iterations.

Case 2: Elastic Testing Data: In order to make the problem more challenging, we then used to model to test the elastic testing data set.

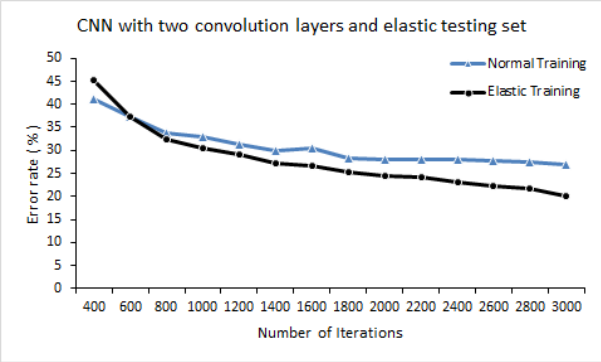

Fig. 8. Convergence for Elastic Testing Data.

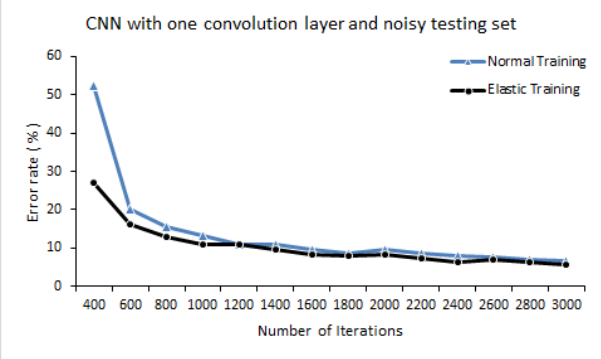

Fig. 9. Convergence for Noisy Testing Data.

The plot above is when we used elastic testing data set and two convolution layers. We can see that the elastic training data set in this case also gives better performance than normal training data set. In addition, the error rate in Fig. 8 is much larger than that in Fig. 7. The reason is that the distortion in testing case badly influenced the recognition, which made the whole accuracy decrease. They became stable almost at the 3000 iterations.

Case 3: Noisy Testing Data: Now we want to see if the elastic distortion technique could partially or totally solve the problem with noisy testing data. So we add some salt and pepper noise in the testing data set.

From Fig. 9 we can see that, with elastic distortion, the CNN can deal with the noise better than normal testing data set. For the convergence rate, they are both stable after about 1500 iterations.

Case 4: Convolution Layers: In this case we compared the results when using different number of convolution layers.

The line with triangle markers in Fig. 10 is the model with 0 convolution layer, which is just the normal neural network (NN). The line with circle markers is the model with 1 convolution layer (CNN-1L) and the line with square markers is the model with 2 convolution layers (CNN-2L). From the large gap between the endpoints of the lines representing NN (triangle) and CNN-1L (circle), we can see the convolution layer actually improved accuracy. However, the endpoints of the lines representing CNN-1L (circle) and CNN-2L (square) are very close, which means the accuracy gained is relatively

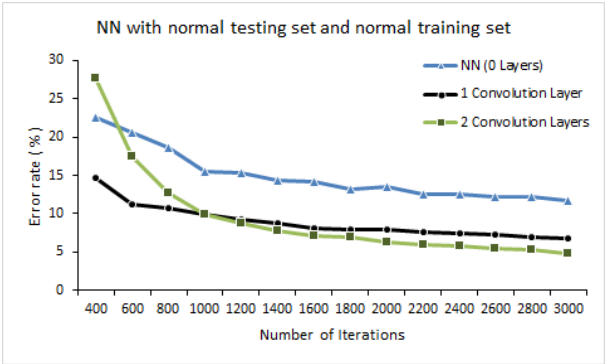

Fig. 10. Convergence for Convolution Layers.

	NN *(0 conv layer)*	CNN-1L *(1 conv layer)*	CNN-2L *(2 conv layers)*
Training Without Distortion	11.72%	6.86%	4.89%
Training With Distortion	12.40%	6.66%	2.53%

Fig. 11. Error rate comparison between elastic training dataset and non-elastic training dataset.

smaller from 1 convolution layer to 2 convolution layers. They converged to the final values after about 2000 iterations. Still, using 2 convolution layers resulted in the lowest error rate.

It is clear by looking at Fig. 11 that the elastic distortion works best when using 2 convolution layers (from the last column), and works second best when using 1 convolution layer (from the second-to-last column).

This trend of improvement shows that the convolution process is able to capture the local feature of each part, including the distortion, while the neural network doesnt have this advantage.

D. Experiment 2

In experiment we did case study about how the CNN model performs. Now we want to see if our distributed computing could indeed improve the efficiency.

Case 1: pseudo distributed computing: Firstly we try to implement the pseudo distributed computing on single machine, this step is to verify that our implementation is correct, which results in two processes simultaneously work to do the training process. Fig. 12 shows the CPU utilization of the two processes on single machine. Since our computation is CPU-based, the CPU utilization is a reasonable measurement to check if these two processes work for our distributed CNN model. Since the machine used here is 4 cores CPU, we can see that the working utilization is near 200% for each process, which means each one takes use of 2 cores.

Case 2: two terminals distributed computing: Now we use two machines to do the parallel computing for the training process. Fig. 13 shows the CPU utilization in this case. We can see that, with each training process in one machine, that process is able to take use of 4 cores in one machine, which means the computation cost would be reduced.

Case 1: 4 terminals distributed computing: Now we try all the 4 machines to do the distributed computing. Fig. 14 shows

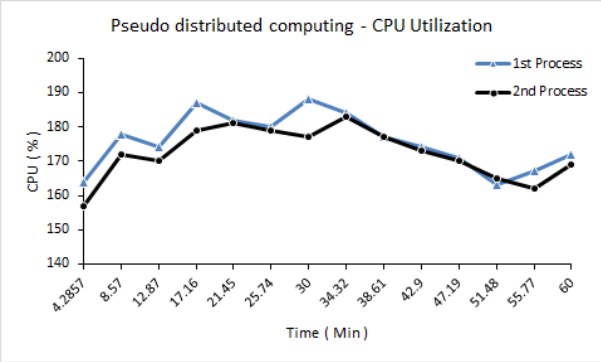

Fig. 12. CPU utilization for Normal Training vs Elastic Training.

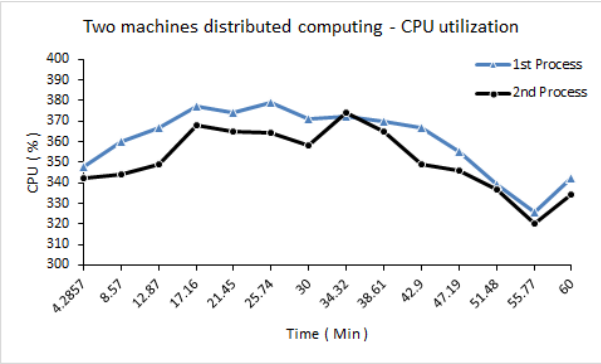

Fig. 13. CPU utilization for Elastic Testing Data.

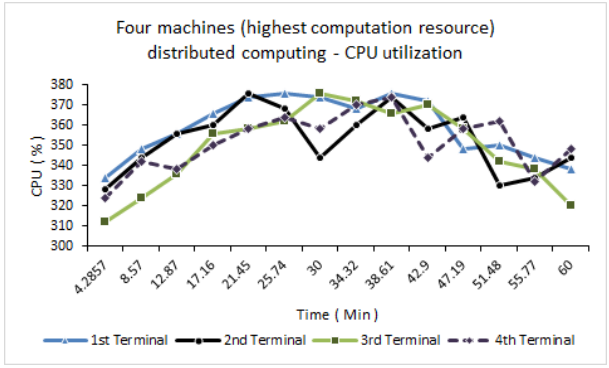

Fig. 14. CPU utilization for Noisy Testing Data.

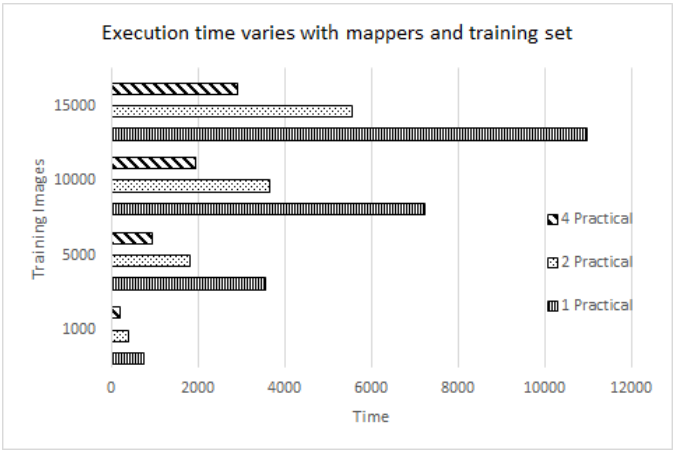

Fig. 15. Execution time with different distribution methods

that all the processes work simultaneously to do the training process.

Besides the cases above, we also change the number of training data in the training process to see the trend of the time consumption.

Fig. 15 shows the execution time with different distributed computing method. The x-axis indicated the time consumption, and the y-axis indicates the number of training data used in the training process.

There are six bars in each of the four cases with different number of training data. The bars with vertical lines at the bottom in each group indicate the pseudo distribute method, the bars with dots in the middle indicate the 2-terminal distributed method, and the bars with the diagonal lines at the top indicate 4-terminal distributed method. Moreover, we use the dark (shaded) bar in each pattern to indicate the practical time cost and the light (non-shaded) bar in each pattern to indicate the theoretical time cost.

With the plot we can see that no matter how many samples we used in the training process, the execution time cannot reach the ideal situation. This extra cost is due to the data communication between terminals, and since our network does not has an extremely fast data communication ability, this would indeed be a problem that undermines the efficiency improvement.

Fig. 16 shows the accuracy performance when we used one or two or four machines after training different number of inputs. In one hand, test accuracy climbs when more training data was involved. It follows the obvious machine learning rule, more training data covering more situations, more precious the model can represent the characteristics of this dataset. The vertical axis indicates accuracy rate while the horizon axis shows how many image used before we got that model. One the other hand, we want to see if using distribution system would have an effect on result performance. Turns out whatever the number of co-work machines, the final results don't change at all. One or two or four machine working at the same time. This figure proves that distribution computation does not change the final result for CNN training problem.

E. Experimental Summary

In summary, we have shown that implementing a distributed model doesn't affect the accuracy of the results - that is, moving from a non-distributed model to a distributed one doesn't negatively impact the accuracy of the result (see Fig. 15) It is also clear from this same figure (Fig. 15) that the more distributed the model, the faster execution time becomes. Going from the non-distributed method to the 2-terminal distributed method execution time is reduced by about 50%; execution time is reduced by a further 50% going from the 2-terminal distribution method to the 4-terminal distributed method.

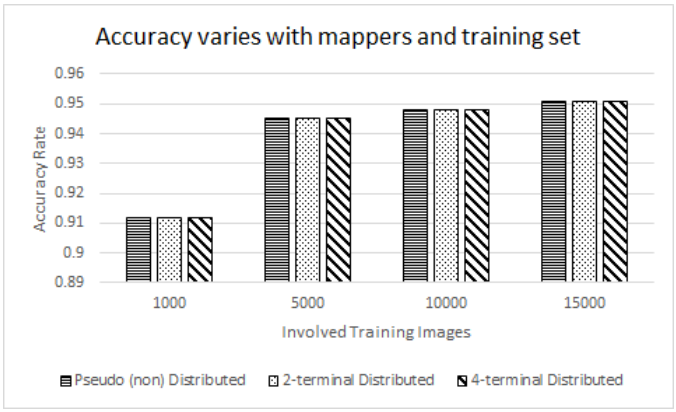

Fig. 16. Accuracy with different distribution methods

Our results yield favorable error rates when utilizing two convolution layers and elastic training (with distortion). Fig. 11 shows that the error rate drops by about 10% when training with distortion and moving from 0 convolution layers (NN) to 2 convolution layers (CNN-2L) (See bottom row of Fig. 11). Even when non-elastic training was used, the error rate drops by about 7% when moving from 0 convolution layers (NN) to 2 convolution layers (CNN-2L). In both cases it is clear the accuracy is improved when increasing from 1 convolution layer (CNN-1L) to 2 convolution layers (CNN-2L).

The results in Fig. 11 are also reflected in Fig. 6 along with error rates when elastic testing was used. With elastic training and normal testing with 2 convolution layers we achieve the lowest error rate (2.53%). This error rate is an improvement upon the case with normal training and normal testing with 2 convolution layers (4.89%). The effectiveness of these experiments using 1 convolution layer did not differ much between elastic and normal training sets, but were worse than the error rates using 2 convolution layers (6.66% instead of 2.53% error rate for elastic training, going from 1 convolution layer to 2 convolution layers; and 6.86% instead of 4.89% error rate for normal training set, going from 1 convolution layer to 2 convolution layers.) Generally the error rates were worse when using elastic testing set due to the increased complexity. Still we achieve about 7% improvement when using 2 convolution layers: from 27.02% error rate using normal training set and elastic testing set to 20.12% error rate using elastic training set and elastic testing set. As expected, he results using elastic training are even worse when using 1 convolution layer: 25.76% using elastic training set and 34.67% using normal training set.

VI. Conclusion

In this research, we used a convolutional neural network to solve the handwritten digit recognition problem. Based on the naive model, we improved the accuracy and the efficiency of this algorithm by implementing two methods: firstly we apply elastic distortion to the training input dataset to help the model better select the feature of the images; Secondly we implemented distributed computing method by mapping the training dataset onto different machines and let them do the training process simultaneously. With both theoretical and empirical analysis, we found that the elastic distortion improves the accuracy by about 7%-10%, and the distributed computing method reduced the training time cost by about 50%.

References

[1] Postal mechanization and early automation. https://about.usps.com/publications/pub100/pub100_042.htm.

[2] H. Barrett. Researching and evaluating digital storytelling as a deep learning tool. In *Society for Information Technology & Teacher Education International Conference*, volume 2006, pages 647–654, 2006.

[3] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001.

[4] X.-W. Chen and X. Lin. Big data deep learning: challenges and perspectives. *Access, IEEE*, 2:514–525, 2014.

[5] J. Dean and S. Ghemawat. Mapreduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[6] D. Gillick, A. Faria, and J. DeNero. Mapreduce: Distributed computing for machine learning. *Berkley, Dec*, 18, 2006.

[7] X. Glorot, A. Bordes, and Y. Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520, 2011.

[8] K. B. Hall, S. Gilpin, and G. Mann. Mapreduce/bigtable for distributed optimization. In *NIPS LCCC Workshop*, 2010.

[9] S. Knerr, L. Personnaz, and G. Dreyfus. Handwritten digit recognition by neural networks with single-layer training. *Neural Networks, IEEE Transactions on*, 3(6):962–968, 1992.

[10] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: A convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113, 1997.

[11] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[12] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60, 1995.

[13] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, volume 1, 2013.

[14] R. P. Lippmann. An introduction to computing with neural nets. *ASSP Magazine, IEEE*, 4(2):4–22, 1987.

[15] A. Morrison, C. Mehring, T. Geisel, A. Aertsen, and M. Diesmann. Advancing the boundaries of high-connectivity network simulation with distributed computing. *Neural computation*, 17(8):1776–1801, 2005.

[16] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[17] X.-X. Niu and C. Y. Suen. A novel hybrid cnn–svm classifier for recognizing handwritten digits. *Pattern Recognition*, 45(4):1318–1325, 2012.

[18] C. Peterson and B. Söderberg. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, 1(01):3–22, 1989.

[19] S. J. Smith, M. O. Bourgoin, K. Sims, and H. L. Voorhees. Handwritten character classification using nearest neighbor in large databases. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(9):915–919, 1994.

[20] K. Sun, X. Wei, G. Jia, R. Wang, and R. Li. Large-scale artificial neural network: Mapreduce-based deep learning. *arXiv preprint arXiv:1510.02709*, 2015.

[21] C. Zanchettin, B. L. D. Bezerra, and W. W. Azevedo. A knn-svm hybrid model for cursive handwriting recognition. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.