Scientific Research Publishing

Journal of Computer and Communications, 2024, 12, 174-190
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Classification with Convolutional Neural Networks in MapReduce

Min Chen

Department of Computer Science, State University of New York at New Paltz, New York, USA
Email: chenm@newpaltz.edu

How to cite this paper: Chen, M. (2024) Classification with Convolutional Neural Networks in MapReduce. *Journal of Computer and Communications*, **12**, 174-190.
https://doi.org/10.4236/jcc.2024.128011

Received: July 18, 2024
Accepted: August 26, 2024
Published: August 29, 2024

Abstract

Deep learning (DL) techniques, more specifically Convolutional Neural Networks (CNNs), have become increasingly popular in advancing the field of data science and have had great successes in a wide array of applications including computer vision, speech, natural language processing, etc. However, the training process of CNNs is computationally intensive and has high computational cost, especially when the dataset is huge. To overcome these obstacles, this paper takes advantage of distributed frameworks and cloud computing to develop a parallel CNN algorithm. MapReduce is a scalable and fault-tolerant data processing tool that was developed to provide significant improvements in large-scale data-intensive applications in clusters. A MapReduce-based CNN (MCNN) is developed in this work to tackle the task of image classification. In addition, the proposed MCNN adopted the idea of adding dropout layers in the networks to tackle the overfitting problem. Close examination of the implementation of MCNN as well as how the proposed algorithm accelerates learning are discussed and demonstrated through experiments. Results reveal high classification accuracy and significant improvements in speedup, scaleup and sizeup compared to the standard algorithms.

Keywords

Distributed System, Image Classification, CNNs, MapReduce, Overfitting

1. Introduction

Image classification is a challenging task and numerous efforts have been made to solve this problem. In recent studies, Deep Learning (DL) techniques presented better predictive performance than traditional machine learning algorithms in many domains, including image classification. Convolutional Neural Networks (CNNs) [1] is a deep learning technique that has gained global recog-

nition in computer vision systems in recent years.

CNN has reached state-of-art performance in a number of applications including image classification [2]-[4], detection [5]-[7] and segmentation [8] [9]. Compared to standard machine learning algorithms, deep learning methods learn from experience and through that, form a hierarchy of concepts to get a better understanding of the world. Learning through experience avoids the need for humans to manually engineer hand-crafted features which is a main advantage of deep models over traditional methods. Additionally, certain deep learning algorithms make explicit assumptions about the input data that allow them to perform remarkably well in specific tasks. For example, CNNs make assumptions about the properties inherent in images such as stationary of statistics and locality of pixel dependencies which allow for fewer connections and parameters, leading to less computational demands and easier training. Furthermore, because the design of CNNs was inspired by the human visual system, they can be trained to recognize complex visual patterns and rich features that have been shown to even outshine human performance.

Convolutional neural networks (CNNs) are similar to traditional artificial neural networks (ANNs). Both perform a feedforward and backward pass which includes computing a linear transformation between the input and weights, followed by a non-linear activation function, calculating the error and then back-propagating the error back to the weights. Both systems are also made up of a series of neurons but the connections between these neurons are how they differ.

In neural networks, each neuron in one layer operates independently and doesn't share any connections but instead was connected to each neuron in the next layer. This is problematic when working with images because a $300 \times 300 \times 3$ image would result in neurons with 270,000 weights which is computationally costly. In addition, the network will also need to be a lot larger to deal with this scale of input. As a result, it is impossible to have unlimited computational power and time to train huge ANNs.

Another reason is that huge ANNs would lead to overfitting. Overfitting occurs when a trained model fits the noise of the data thus hampering its ability to generalize well to new examples. Reducing the complexity of ANNs can stop or reduce the effects of overfitting. The less parameters require to train, the less likely the network will overfit to improve the predictive performance of the model.

Luckily, CNNs specialize in dealing with images by utilizing convolution along with local receptive fields to help control the amount of parameters. Imagine taking a filter of size 5×5, and sliding it across an entire image to find features that match the filter. This is exactly how convolution is performed but the filter is initialized with weights and dot products are computed between the weights and input image exactly as in neural networks. With convolution, instead of neurons being connected in a fully connected manner, neurons in one layer are only connected to a small region of the layer before it which is controlled by the size of the filter or receptive field.

Despite these remarkable qualities, a major drawback with deep learning models is that they require large amounts of data in order to perform well. Only through the rise of large annotated datasets such as ImageNet [4] and the growth of graphics processor units (GPUs) have CNNs been able to achieve the recent success that they've had. Additionally, with the rise of big data and increased complexity of tasks, training CNNs incur a high computational cost which is infeasible on a single machine without large computational resources. Moreover, CNN architectures have become increasingly deeper which on one hand, allows for better feature representations but also increases the complexity of the network. ResNet [10] is about twenty times deeper than AlexNet [4] and eight times deeper than VGGNet [3]. State-of-art CNNs require a massive number of parameters that need to be tuned during training which leads to extensive training times and models that are highly difficult to optimize.

To cope with increasingly complex tasks and volumes of data, the evolution of CNNs has severely impacted their efficiency. CNNs are innately both data and computationally intensive which makes speed and storage capacity a large limiting factor in reaching performance and scalability requirements. To overcome the imposed time and space obstacles, this work implements a parallelized CNN algorithm based on MapReduce [11] (MCNN) on a cloud computing cluster. The developed algorithm takes advantage of the computational structures inherent in CNNs that lends them to parallelization to achieve increased processing speed. Additionally, the use of cloud computing provides an economical means to facilitate data intensive applications by providing workload balancing and resource scheduling.

The rest of the paper is organzied as follows. Section 2 reviews some related CNN work in literature. Section 3 briefly introduces CNN algorithm. Section 4 describes in detail the design and implementation of the distributed MCNN algorithm. Section 5 evaluates the performance of the proposed MCNN in a MapReduce environment. Section 6 concludes the paper and points out some future work.

2. Related Work

Due to the shortcomings of the traditional process, deep learning algorithms, more specifically CNN, are widely used for image classification. Commonly image processing tasks successfully addressed by CNN are handwritten and image recognition problems [12].

Besides the handwritten problem, [13] uses a variation of convolutional networks, namely Neocognitron (NEO), for face recognition task. The positive rates for the NEO decline significantly when the classifier is tested under more unconstrainted conditions. In [14], a visual automated system using CNN was adopted for visual tunnel inspection. A deep convolutional networks (ConvNets) is adopted to localize wooden knots in images of oak board. A significant improvement has been found by comparing to support vector machine. As in other studies, CNN outperformed the traditional machine learning techniques in [15].

This study investigated CNN applied to defect detection in different materials. By comparing to other studies, CNN does not need a feature extraction process due to its embedded module.

A data mining approach is proposed in [16] to improve the local binary patterns in texture analysis. Three different descriptors with three spatial resolutions are used to evaluate the proposed approach in texture images. In [17], a J48 classifier is built to evaluate colonoscopy exam images using texture descriptors. As shown in the results, the proposed classifier has achieved a relatively high sensitivity. In [18], a CNN with combination of texture-based feature extraction techniques are used for biological image classification. The proposed algorithm is compared with traditional techniques such as decision tree, neural networks, nearest neighbors and support vector machine. The proposed CNN algorithm has achieved predictive performance superior to traditional classification techniques.

For image classification through MapReduce, the literature presents the improvement of statellite images in [19]. A system based on Hadoop that implements the MapReduce programming model is used to improve the classification of large-scale remote sensing images. In [20], a MapReduce-based distributed SVM is used for image classification annotation. SVM with bagging have shown better performance in classification than a single SVM. The proposed algorithm re-samples the training dataset based on bootstrapping. The training time is reduced significantly with a high level of accuracy in classification. A parallel design and realization method for particle swarm optimization with back-propagation neural network is proposed in [21] to improve the classification accuracy and runtime efficiency of the back-propagation neural network. The results demonstrate both higher accuracy and improved time efficiency from applying parallel processing on big data.

To summarize, research on CNN algorithms has been carried out from various dimensions, but mainly focuses on improving the classification accuracy. Improving the runtime efficiency of a CNN still remains an open challenge. This motivates the design of a Mapreduce-based CNN, which is an efficient distributed CNN algorithm building on a highly scalable MapReduce implementation for image classification.

3. Convolutional Neural Networks

Convolutional neural networks are comprised of three types of layers: convolutional layers, pooling layers and fully-connected layers.

3.1. Convolutional Layer

The convolutional layer determine the output of neurons. These neurons are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume. The convolutional layer keep the local connection weights fixed for the next layer in order to reduce the size of parameters. It provides an opportunity to detect

and recognize features regardless of their positions in the image.

There exists several hyper-parameters (values set before training) that can be adjusted when performing convolution: the filter size, padding and stride are three. First, the size of the receptive field controls the connection of neurons to their input spatially (width and height) but always through the entire depth. For example, using a receptive field of size 5 × 5 on a 300 × 300 × 3 image leads to neurons with $5\times5\times3=75$ weights. A large receptive field may capture more context but could also lead to a loss in finer details. Secondly, notice how in Figure 1 the border of the input is padded with zeros. This is set to preserve the size of the input and output so that additional convolutions will not cause the input to shrink too quickly, resulting in information loss. Lastly, the stride determines how many pixels to move when the filter is slid across the image; a stride of one or two is commonly used in practice.

The output size after convolution can be computed by the function:

$$(W-F+2P)/S+1 \tag{1}$$

where W is the input size, F the filter size, S the stride and P the amount of padding used. For example with an input (W) = 5 × 5, filter size (F) = 3 × 3, stride (S) = 2, padding (P) = 1 then the output size = 3 × 3.

Just as with neural networks, after a linear transformation is applied between the input and weights, an activation function is used. The ReLU (Rectified Linear Unit) [22] is a common activation function used with CNNs. It simply computes $f(x)=\max(0,x)$ which removes negative values by placing a threshold at zero. See Figure 1 for an illustration.

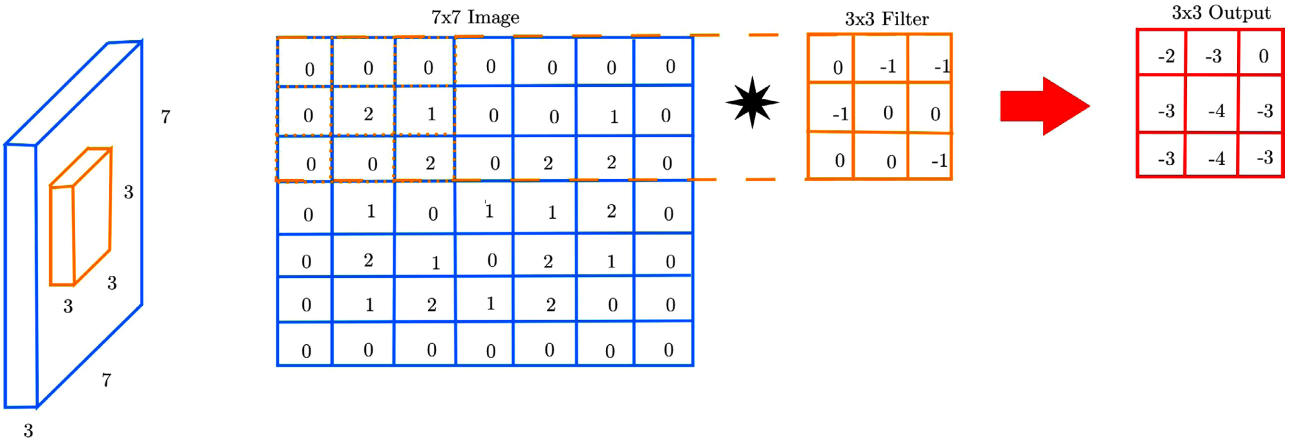

Figure 1. Convolution performed on a single channel of an image. A filter is slid across the entire input, computing an element-wise product between each input pixel and the filter then summing the results to form the output elements. Here *padding* = 1 and *stride* = 2.

3.2. Pooling Layer

The aim of pooling layer is to gradually reduce the dimensionality of the representation in order to further reduce the number of parameters and the computational complexity of the model. General pooling layers are consist of pooling

neurons that are able to perform a multitude of common operations including LeRu normalization, and average pooling. The pooling layer operates over each activation map in the input and scales its dimensionality using a "MAX" function. This is also called max-pooling layer. Because of the destructive nature of the pooling layer, the stride and filters of the pooling layers are two generally observed methods of max pooling. Both are set to 2×2 to allow the layer to extend through the entirety of the spatial dimensionality of the input.

Pooling layers act as a way to reduce the size of the input, number of parameters and computational cost; it also helps control overfitting. The most commonly used pooling layer is max pooling which takes the max element of a region in the input and discards the rest. Figure 2 shows an example of the pooling layer.

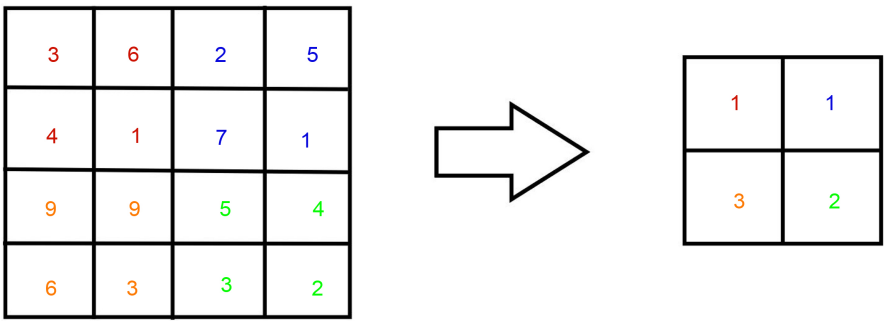

Figure 2. Max pooling operation performed using a filter of size 2×2 and *stride* = 2.

3.3. Fully-Connected Layer

The fully-connected layer is analogous to the way that neurons are arranged in traditional forms of ANNs. It contains neurons of which are directly connected to the neurons in the two adjacent layers.

Convolutional neural networks are essentially a series of convolution and pooling layers stacked together with a fully connected layer inserted before the output layer. Additional layers help the network learn more complex features. For example, if the network is fed the image of a face then the first layer will detect low-level features such as edges while deeper layers detect higher level features such as eyes or a nose and eventually an entire face.

An example of a stack of layers for CNN is shown in Figure 3.

4. The Design of MapReduce-Based CNN (MCNN)

Convolutional Neural Networks is one deep learning algorithm that can benefit from the parallelized computation offered by the MapReduce programming model. CNN iteratively adjusts weights in the network by computing their partial gradients after each set of the training data is propagated through the network. Thus parallelization during the training phase can be accomplished by distributing the data into a number of chunks. Each data chunk can then be fed to several CNNs and each CNN can be trained independently in parallel. The

outputs can then be aggregated to produce the final results which are then used to update the weights for the next iteration. Figure 4 below shows a high level overview of the procedure.

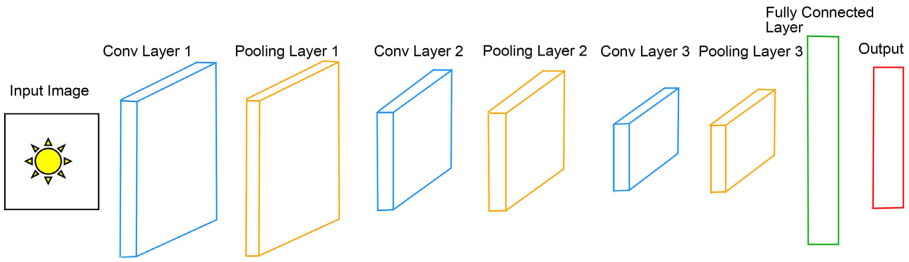

Figure 3. An example of CNN architecture.

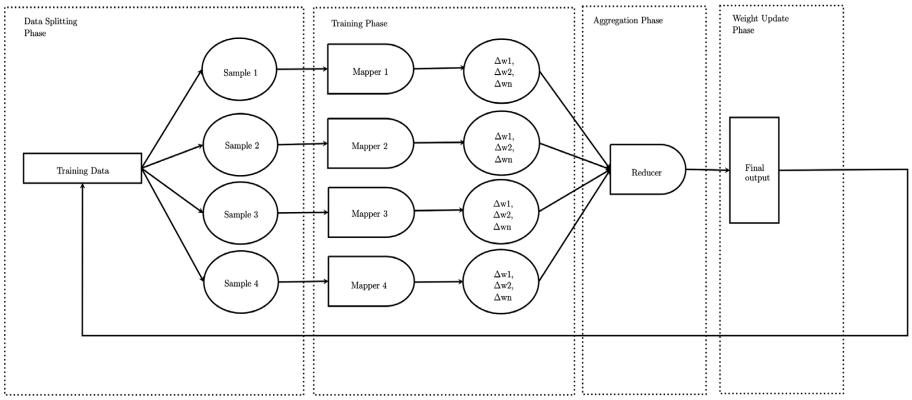

Figure 4. A flowchart of the proposed distributed CNN.

The mappers take as input with the training data (x,y) and a set of randomly initialized weights $[w_1, w_2, \cdots, w_n]$. The set of weights contain n weights to represent the number of hidden layers in the network so w_i corresponds to the weights for hidden layer i. The training data is represented as a set of tuples (x,y) where x is a training instance and y is the ground truth label for that instance. Each mapper initializes the network with the given set of weights and the network is trained using the input samples. The output is a set of newly trained *updated Weights* $[\Delta w_1, \Delta w_2, \cdots, \Delta w_n]$ which is then fed to the reducer. After each iteration, the mappers receive a set of updated weights which are processed through the network until the max number of iterations is reached. The pseudo code for the map function is shown in **Algorithm 1**.

Algorithm 1 Map Function

Input: A set of tuples (x, y), where x is a training sample and y is the ground truth. A list of randomly initialized weights for each hidden layer
Output: Updated weights
procedure MAP(Data)
    Read weights from HDFS
    Initialize CNN with random weights
    for i = 0 to numLayers - 1 **do**
        updatedWeights[i] = updatedWeights[i] + Weights[i]
    emit updatedWeights

The reducer receives as input the intermediate output by the mappers and ag-

gregates the weights produced from each of the mappers. Since the position of weights in the output of mappers are sorted according to the hidden layer they belong to, the reducer can simply aggregate weights according to their index. The aggregation computes a cumulative sum over weights and then divides by the number of training instances in the batch to form an average of weights. The final result is used to update the weights in the network and sent to the mapper for the next iteration. The pseudo code for the reduce function is shown in **Algorithm 2**.

Algorithm 2 Reduce Function

```
Input: Updated weights
Output: Accumulated weights
procedure REDUCE(updatedWeights)
    sumWeights = Initialize list of zeros;
    for i = 0 to numLayers-1 do
        sumWeights[i] = sumWeights[i] + Weights[i]
    output: sumWeights
```

MapReduce jobs utilize a driver to serve as the scheduler for tasks. The driver creates a directed acyclic graph (DAG) or execution plan for the program which are then divided into smaller tasks to be executed. communicates with Hadoop, determines which map and reduce classes are used and specifies the configuration of jobs. Configurations are provided by the user and include the path to the training data, path of the output, training parameters and configurations for the network. Training parameters include number of training samples, number of validation samples, max number of iterations, max epochs and batch size. Network configurations include the size of the receptive field, stride, number and type of layers, learning rate and optimization method. The pseudo code for the driver function is shown in **Algorithm 3**.

Algorithm 3 Driver Function

```
procedure DRIVER
    for layer = 0 to numLayers-1 do
        Weights[layer] = Randomly Initialized Weights
    numIteration = 1
    while numIteration <= maxIterations do
        Initialize MapReduce Job;
        Store weights in HDFS;
        Pass model parameters to MapReduce Job;
        Run MapReduce jobs:
        Input for Mappers: Weights for each hidden layer;
        Mappers compute updated weights for every hidden layer;
        Output of Mappers: Newly trained weights;
        Aggregate Weights of each hidden layer:
        Input to Reducer: Newly trained weights from Mappers;
        Reducers aggregate weights for each hidden layer:
        for layer = 0 to numLayers-1 do
            Weights[layer] =
            Weights[layer] + updatedWeights[layer]
        Output of Reducer: Cumulative sum of weights for each hidden layer;
        Update weights of each hidden layer;
     numIteration++;
    output: Final Weights
```

5. Experiments and Results

5.1. MCNN Architecture and Parameters

The proposed MCNN architecture used in experiments is an adaptation of VGGNet. The input is 32 × 32 RGB images with 49,000 images used for training and 1000 for validation. The number of training iterations is set at a maximum of 60 epochs. No data augmentation is used, training data mean subtraction is the only pre-processing step. See Table 1 for more details on training parameters.

The network contains 10 weighted layers: 8 convolutional (conv.) layers and 2 fully-connected (FC) layers. The number of channels begins with 64 in the first layer and doubles after every max-pooling layer until it reaches 512 in the last convolution layer. Small 3 × 3 receptive fields and stride 1 are used throughout the network. Every conv. layer is followed by the ReLU non-linearity. The initialization [23] is used at every weighted layer. Two fully-connected layers are inserted after the stack of convolutional layers. The first fully-connected layer contains 500 channels and the second contains 10 channels for the 10 classes contained in the CIFAR-10 dataset. The final layer is the soft-max layer to obtain class probabilities and categorical output. Configuration details are outlined in Table 2.

Table 1. CNN Architecture details. The convolutional layer parameters are denoted as conv<receptive field size> - <number of channels>.

Input (32 × 32 RGB image)
conv3-64
conv3-64
maxpool
conv3-128
conv3-128
maxpool
conv3-256
conv3-256
maxpool
conv3-512
conv3-512
maxpool
FC-500
FC-10
soft-max

Table 2. Model parameters.

Train samples	Validation samples	Batch size	Max Epochs	Optimizer	Learning Rate
49,000	1000	100	60	Adam	0.0001

5.2. Distributed Computing Environment

The algorithm is deployed on a Hadoop cluster using Amazon Web Service EMR which provides economical, large capacity, remote computing services. Parallel MCNN is implemented using Spark, an extension of the MapReduce framework which supports fast in memory computations, specifically designed for iterative machine learning algorithms. The proposed parallel algorithm is run using the following distributed computing environment:

- Hadoop: The cloud compute cluster to assign namenode and datanodes, conduct work load balancing, resource scheduling and data replication.
- HDFS: The distributed file system that provides fault tolerance and high throughput access to large datasets.
- YARN: Provides job scheduling and resource management.
- SPARK: An extension of the MapReduce framework which makes use of Resilient Distributed Datasets (RDDs) as a fault tolerant data structure operated on in parallel.

Table 3 provides more details on the cluster specifications.

Table 3. Cluster details.

Namenode	Datanodes	Bandwidth	Hadoop	Spark
CPU: E5@2.4 GHz Memory: 32 GB OS: Linux	CPU: E5@2.4 GHz Memory: 32 GB OS: Linux	1 Gbps	2.8.4	2.3.1

5.3. Experimental Results

Experiments conducted to evaluate the performance of the proposed MCNN include measurements of accuracy, speedup, scaleup and sizeup. The dataset used in all experiments come from the CIFAR-10 [24] dataset which contains 50,000 RGB images of size 32×32.

5.3.1. MCNN with or without Dropout

The initial proposed algorithm, MCNN1, achieves 89% accuracy on both training and validation data. However, overfitting is an issue using the MCNN1 as seen by the gap between the training and validation curves in Figure 5 and Figure 6, respectively. In order to overcome this problem, dropout [25] layers are introduced and added after every max pool layer and after the first fully connected layer. A dropout value of 0.4 was used throughout the network. The modified version of the proposed algorithm, namely MCNN2, shows a better performance of the network after adding dropout. The classification accuracies for training and validation data have improved significantly.

Figure 7 and Figure 8 compare the effect of dropout on both training and validation accuracy. Dropout essentially adds noise to the network by ignoring a fraction of nodes during training which reduces co-adaptation of neurons. Thus neurons are forced to learn independently and not over rely on one another. The effect is shown by the wider spread loss curve in Figure 6 and lower training ac-

curacy shown in Figure 7. But dropout provides the added benefit of improved generalization as shown from the higher validation accuracy achieved in Figure 8. Furthermore, Figure 8 reveals that dropout requires a greater number of training epochs to take effect. From epochs 0 to 20 the network without dropout consistently receives higher validation accuracy. Between 20 to 30 epochs, both networks are effectively equal in accuracy but not until after 30 epochs is when dropout reveals its performance boost.

Figure 5. Training loss without Dropout.

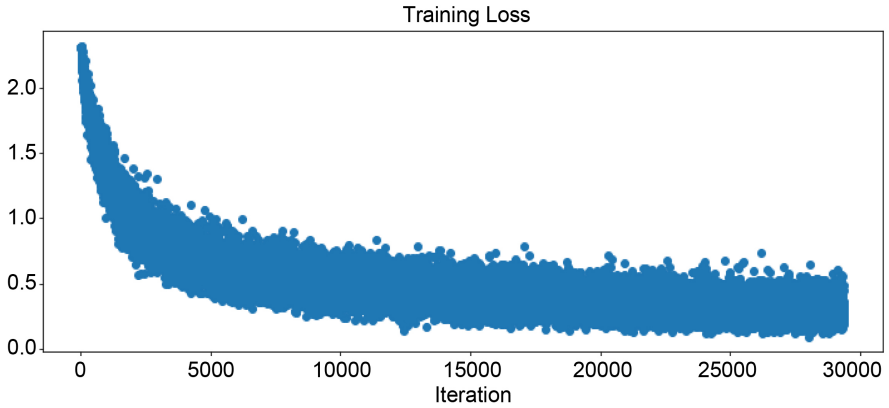

Figure 6. Training loss with Dropout.

5.3.2. Scalability of the Proposed MCNN

Improvements in speed and scalability of the proposed algorithm is also measured. Speedup is defined as the ability for a system to yield m times speedup with m times the number of nodes. To measure speedup the number of samples were kept constant at 50,000 while the number of nodes were increased by 2, 4, 8, 16, 32, and 64 respectively. For each number of nodes, five trials were conducted and the average time was recorded. Results are shown in Figure 9 where the blue line represents MCNN1, which is the proposed MCNN without dropout layers. The red line represents the MCNN2 which is the proposed MCNN with dropout layers. The parallel algorithm achieves close to linear speedup in MCNN1 and MCNN2. Exact linearity isn't achieved due to the communication overhead with a large number of nodes. Also, the MCNN2 perform better than MCNN1 in

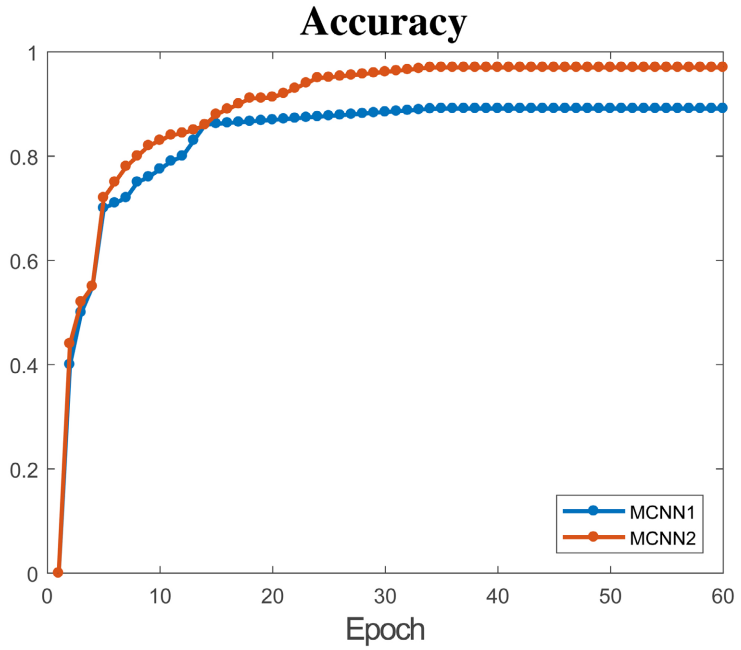

Figure 7. Train accuracy with MCNN1 and MCNN2.

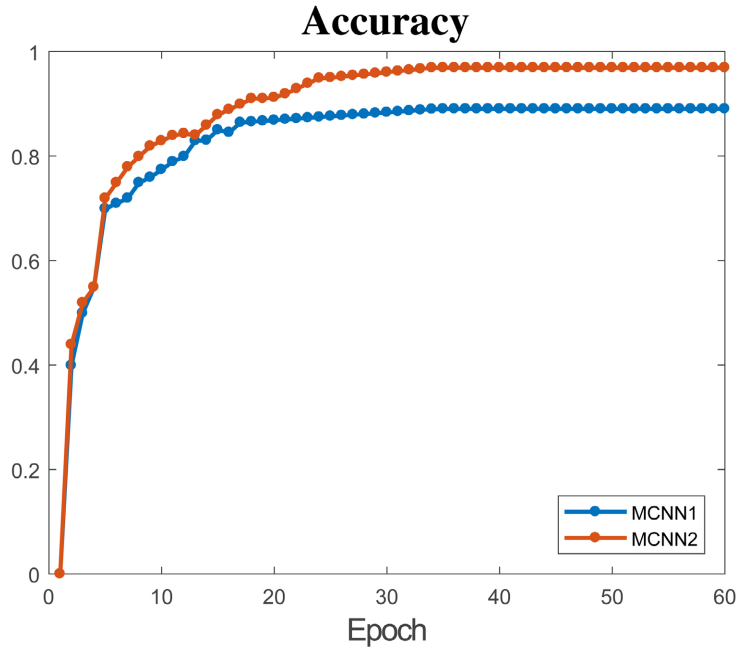

Figure 8. Validation accuracy with MCNN1 and MCNN2.

terms of speed up due to the reduce number of neurons used in the networks.

Scaleup measures the ability of a m-times larger system to perform a m-times larger task at the same time as the original system. To evaluate scaleup the dataset size is increased in proportion to the number of nodes in the system. The number of samples is increased from 10 k, 20 k, 30 k, 40 k and the number of nodes is increased from 1 to 4. Five trials for each sample size was conducted and the average time was recorded. Figure 10 depicts the results of the scaleup

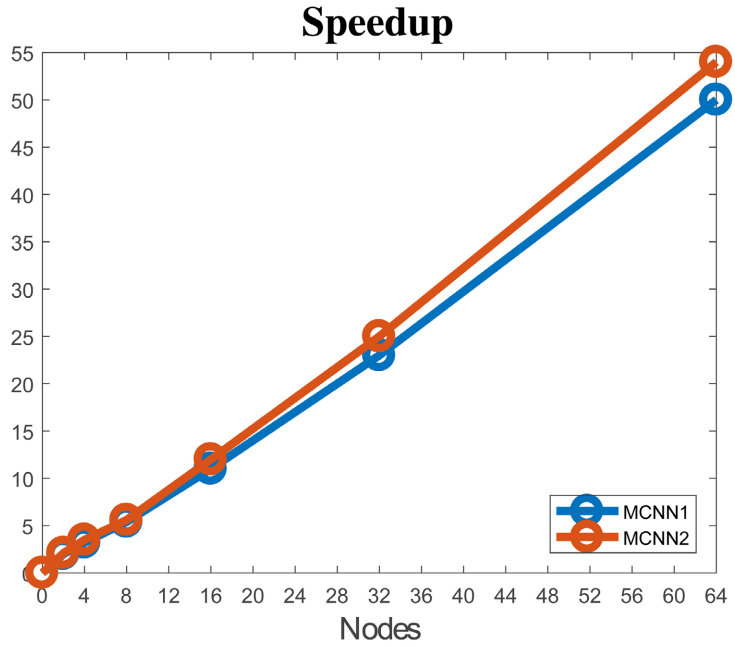

Figure 9. Speedup of MCNN1 and MCNN2.

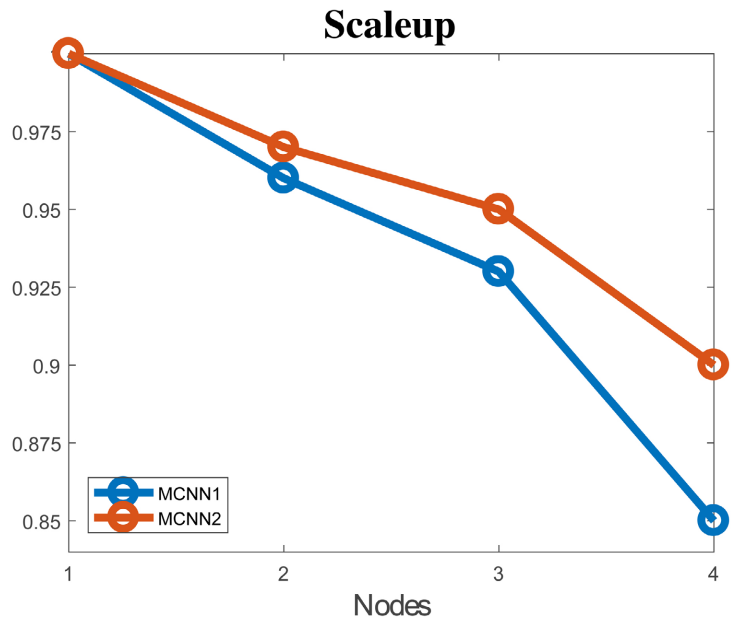

Figure 10. Scaleup of MCNN1 and MCNN2.

experiments. Again, MCNN2 perform better than MCNN1 as the number of nodes increases. Results reveal that scaleup improves as the number of nodes in the system increases.

Sizeup increases the size of the input data by a factor of m while holding the number of machines in the system constant. To measure sizeup, experiments are run on 1, 2 and 4 machines respectively. For each number of machines, the dataset size is increased from 10 k, 20 k, 40 k, 80 k, and 160 k samples. The results in Figures 11-13 show the proposed algorithm performs well in terms of sizeup

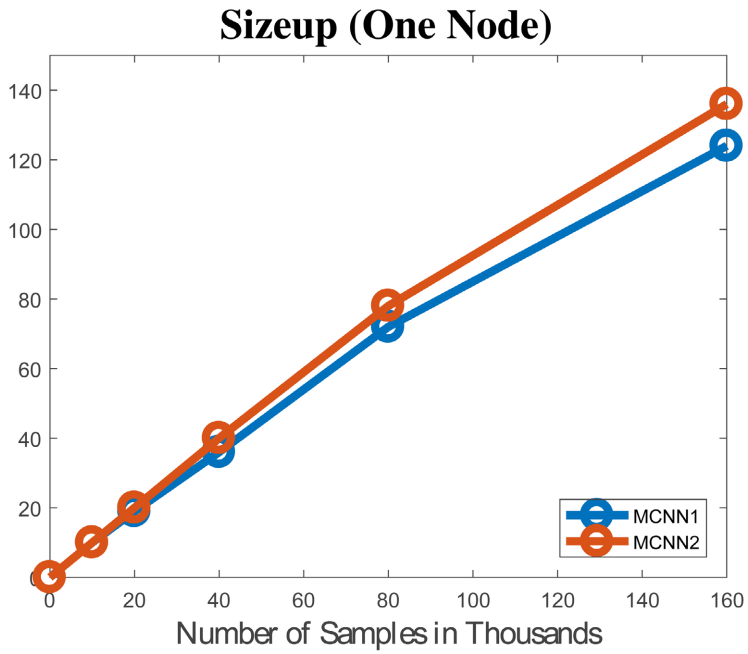

Figure 11. Sizeup of MCNN1 and MCNN2 using one node.

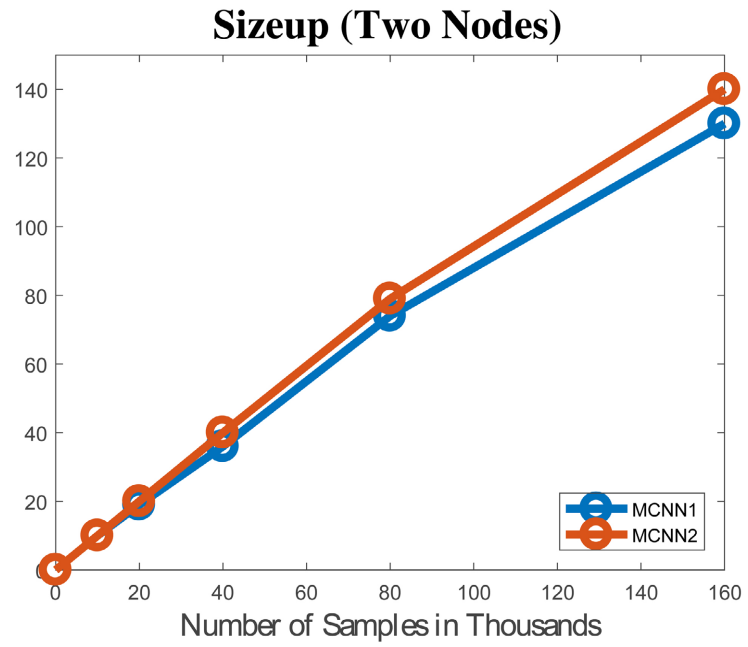

Figure 12. Sizeup of MCNN1 and MCNN2 using two nodes.

but that as the number of machines increases, sizeup performance is hampered due to the communication overhead between nodes.

Overall, the MCNN2 performs better than MCNN1 in speedup, scaleup and sizeup. However, as the number of nodes increases, the linearity decreases due to the communication overhead of Hadoop clusters. But MapReduce framework is still a good approach in deep learning as the significantly reduction of computation and the use of cheap commodity computers.

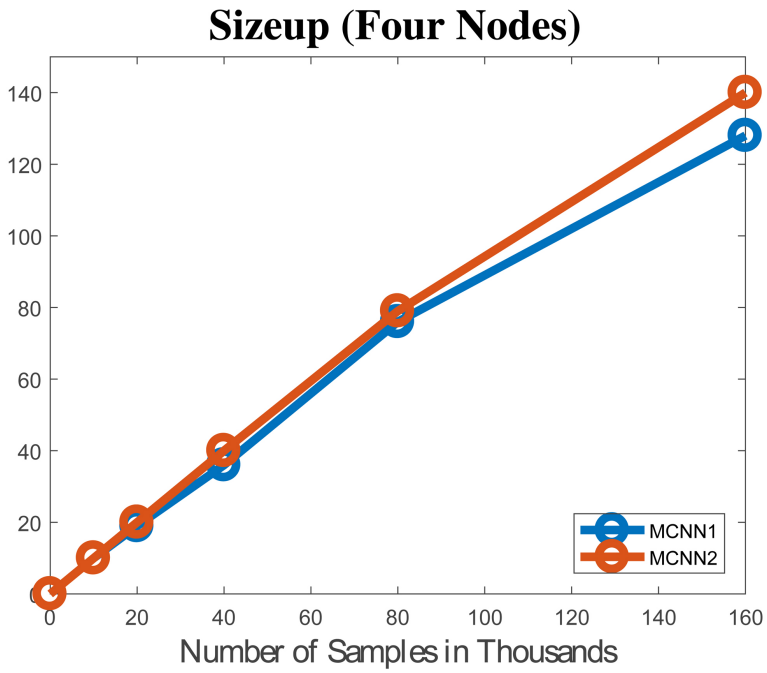

Figure 13. Sizeup of MCNN1 and MCNN2 using four nodes.

6. Conclusions

With the rise of big data and increased complexity of tasks, the efficiency of deep learning algorithms have been severely impacted by the nature of long training times and high computational cost. To be able to solve even greater problems of the future, learning algorithms must maintain high speed and accuracy through economical means. MapReduce [11] is one of the most efficient big data solutions, which enables the processing of a massive volume of data in parallel with many low-end computing nodes. This programming paradigm is a scalable and fault-tolerant data processing tool that was developed to provide significant improvements in large-scale data-intensive applications in clusters. To that end, this paper takes advantage of the MapReduce framework to develop a parallel CNN algorithm (MCNN). The proposed MCNN achieves high classification accuracy in image classification and close to linear speedup, scaleup and sizeup. The results demonstrate that the MapReduce framework is an effective tool to improve the speed and scalability of CNNs.

In terms of directions for future work, several ideas come to mind. Experiments conducted here utilize images from CIFAR-10. A much larger dataset such as ImageNet with additional compute nodes could be used to analyze how this affects performance in speedup, scaleup and sizeup. Additionally, instead of image classification, other computer vision tasks such as image segmentation or object detection could be applied. Trying other CNN architectures such as ResNet or GoogLeNet [26] and examining their performance in accuracy and speed of convergence is another interesting path. Lastly, ensemble methods similar to the work done in [27] can also be explored.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, **86**, 2278-2324. https://doi.org/10.1109/5.726791

[2] Egmont-Petersen, M., de Ridder, D. and Handels, H. (2002) Image Processing with Neural Networks—A Review. *Pattern Recognition*, **35**, 2279-2301. https://doi.org/10.1016/s0031-3203(01)00178-9

[3] Simonyan, K. and Zisserman, A. (2014) Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv: 1409.1556. https://doi.org/10.48550/arXiv.1409.1556

[4] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., *et al.* (2015) ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, **115**, 211-252. https://doi.org/10.1007/s11263-015-0816-y

[5] Nowlan, S.J. and Platt, J.C. (1994) A Convolutional Neural Network Hand Tracker. *Proceedings of the 7th International Conference on Neural Information Processing Systems*, Denver Colorado, 1 January 1994, 901-903.

[6] Everingham, M., Eslami, S.M.A., Van Gool, L., Williams, C.K.I., Winn, J. and Zisserman, A. (2014) The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, **111**, 98-136. https://doi.org/10.1007/s11263-014-0733-5

[7] Lin, T., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., *et al.* (2014) Microsoft COCO: Common Objects in Context. *Computer Vision—ECCV* 2014, Zurich, 6-12 September 2014, 740-755. https://doi.org/10.1007/978-3-319-10602-1_48

[8] Ciresan, D., Giusti, A., Gambardella, L. and Schmidhuber, J. (2012) Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. *Advances in Neural Information Processing Systems*, **25**, 2843-2851.

[9] Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014) Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 *IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, 23-28 June 2014, 580-587. https://doi.org/10.1109/cvpr.2014.81

[10] He, K., Zhang, X., Ren, S. and Sun, J. (2016) Deep Residual Learning for Image Recognition. 2016 *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), Las Vegas, 27-30 June 2016, 770-778. https://doi.org/10.1109/cvpr.2016.90

[11] Leung, J. and Chen, M. (2019) Image Recognition with MapReduce Based Convolutional Neural Networks. 2019 *IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference* (*UEMCON*), New York, 10-12 October 2019, 119-125. https://doi.org/10.1109/uemcon47517.2019.8992932

[12] Cun, Y.L., Boser, B., Denker, J.S., Henderson, D., Howar, R.E., Hubbard, W. and Jackel, L.D. (1990) Handwritten Digit Recognition with a Back-Propagation Network. *Advances in Neural Information Processing Systems*, **160**, 396-404.

[13] Nebauer, C. (1998) Evaluation of Convolutional Neural Networks for Visual Recognition. *IEEE Transactions on Neural Networks*, **9**, 685-696. https://doi.org/10.1109/72.701181

[14] Norlander, R., Grahn, J. and Maki, A. (2015) Wooden Knot Detection Using Con-

vnet Transfer Learning. *Image Analysis*, Copenhagen, 15-17 June 2015, 263-274. https://doi.org/10.1007/978-3-319-19665-7_22

[15] Park, J., Kwon, B., Park, J. and Kang, D. (2016) Machine Learning-Based Imaging System for Surface Defect Inspection. *International Journal of Precision Engineering and Manufacturing-Green Technology*, **3**, 303-310. https://doi.org/10.1007/s40684-016-0039-x

[16] Kwak, J.T., Xu, S. and Wood, B.J. (2015) Efficient Data Mining for Local Binary Pattern in Texture Image Analysis. *Expert Systems with Applications*, **42**, 4529-4539. https://doi.org/10.1016/j.eswa.2015.01.055

[17] Oliva, J.T., Lee, H.D., Spolaôr, N., Coy, C.S.R. and Wu, F.C. (2016) Prototype System for Feature Extraction, Classification and Study of Medical Images. *Expert Systems with Applications*, **63**, 267-283. https://doi.org/10.1016/j.eswa.2016.07.008

[18] Affonso, C., Rossi, A.L.D., Vieira, F.H.A. and de Leon Ferreira de Carvalho, A.C.P. (2017) Deep Learning for Biological Image Classification. *Expert Systems with Applications*, **85**, 114-122. https://doi.org/10.1016/j.eswa.2017.05.039

[19] Chebbi, I., Boulila, W. and Farah, I.R. (2016) Improvement of Satellite Image Classification: Approach Based on Hadoop/MapReduce. 2016 2*nd International Conference on Advanced Technologies for Signal and Image Processing* (*ATSIP*), Monastir, 21-23 March 2016, 31-34. https://doi.org/10.1109/atsip.2016.7523046

[20] Alham, N.K., Li, M., Liu, Y. and Hammoud, S. (2011) A Mapreduce-Based Distributed SVM Algorithm for Automatic Image Annotation. *Computers & Mathematics with Applications*, **62**, 2801-2811. https://doi.org/10.1016/j.camwa.2011.07.046

[21] Cao, J., Cui, H., Shi, H. and Jiao, L. (2016) Big Data: A Parallel Particle Swarm Optimization-Back-Propagation Neural Network Algorithm Based on Mapreduce. *PLOS ONE*, **11**, e0157551. https://doi.org/10.1371/journal.pone.0157551

[22] Nair, V. and Hinton, G.E. (2010) Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the* 27*th International Conference on International Conference on Machine Learning*, Haifa, 21-24 June 2010, 807-814.

[23] He, K., Zhang, X., Ren, S. and Sun, J. (2015) Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. 2015 *IEEE International Conference on Computer Vision* (*ICCV*), Santiago, 7-13 December 2015, 1026-1034. https://doi.org/10.1109/iccv.2015.123

[24] Alex, K. (2009) Learning Multiple Layers of Features from Tiny Images. https://www.cs.toronto.edu/kriz/learning-features-2009-TR.pdf

[25] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, **15**, 1929-1958.

[26] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., *et al.* (2015) Going Deeper with Convolutions. 2015 *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), Boston, 7-12 June 2015, 1-9. https://doi.org/10.1109/cvpr.2015.7298594

[27] Liu, Y., Yang, J., Huang, Y., Xu, L., Li, S. and Qi, M. (2015) MapReduce Based Parallel Neural Networks in Enabling Large Scale Machine Learning. *Computational Intelligence and Neuroscience*, **2015**, Article 297672. https://doi.org/10.1155/2015/297672