Postprint: Parallel Deep Convolutional Neural Network Optimization Algorithm Based on Im2col

Authors: Hu Jian, GONG Ke, Mao Yimin, Chen Zhigang, Chen Liang

Date: 2022-06-06T00:00:00+00:00

Abstract

To address the issues of excessive redundant features, slow convolutional layer computation, and poor loss function convergence in parallel deep convolutional neural network (DCNN) algorithms within big data environments, this paper proposes an Im2col-based parallel deep convolutional neural network optimization algorithm, IA-PDCNNOA. First, a Marr-Hildreth operator-based parallel feature extraction strategy (MHO-PFES) is proposed to extract target features from the data as input for the convolutional neural network, effectively avoiding the problem of excessive data redundancy. Second, an Im2col-based parallel model training strategy (IM-PMTS) is designed, which removes redundant convolution kernels by designing a Mahalanobis distance-based center value and combines MapReduce with the Im2col method for parallel model training, thereby improving convolutional layer computation speed. Finally, an improved mini-batch gradient descent strategy (IM-BGDS) is proposed to exclude the influence of abnormal node training data on batch gradients, thus solving the problem of poor loss function convergence. Experimental results demonstrate that the IA- PDCNNOA algorithm exhibits favorable performance for deep convolutional neural network computation in big data environments and is suitable for parallelized deep convolutional neural network model training on large-scale datasets.

Full Text

Preamble

Vol. 39 No. 10
Application Research of Computers (ChinaXiv Cooperating Journal)
Accepted Paper

Parallel Deep Convolution Neural Network Optimization Based on Im2col

HU Jian[1],[2], GONG Ke[1], MAO Yimin[1]†, CHEN Zhigang[3], CHEN Liang[2]

[1] School of Information Engineering, Jiangxi University of Science & Technology, Ganzhou, Jiangxi 341000, China

[2] Electronic Information Engineering, Gannan University of Science & Technology, Ganzhou, Jiangxi 341000, China

[3] College of Computer Science & Engineering, Central South University, Changsha 410083, China

Abstract

In large-scale data environments, parallel deep convolutional neural network (DCNN) algorithms suffer from excessive data redundancy, slow convolutional layer operations, and poor loss function convergence. This paper proposes a parallel deep convolutional neural network optimization algorithm based on the Im2col method, called IA-PDCNNOA. First, we introduce a parallel feature extraction strategy based on the Marr-Hildreth operator (MHO-PFES) to extract target features from data as input to the convolutional neural network, effectively avoiding excessive data redundancy. Second, we design a parallel model training strategy based on the Im2col method (IM-PMTS), which removes redundant convolution kernels by designing Mahalanobis distance center values and combines MapReduce with Im2col for parallel model training, thereby improving convolutional layer operation speed. Finally, we propose an improved mini-batch gradient descent strategy (IM-BGDS) that eliminates the influence of abnormal node training data on batch gradients, solving the problem of poor loss function convergence. Experimental results demonstrate that the IA-PDCNNOA algorithm exhibits strong performance in deep convolutional neural network computation under big data environments and is suitable for parallelized deep convolutional neural network model training on large-scale datasets.

Keywords: big data; DCNN algorithm; parallel computing; feature extraction; image classification

0 Introduction

Deep Convolutional Neural Networks (DCNN) represent a crucial class of classification algorithms in deep learning, possessing powerful representation, generalization, and fitting capabilities. They deliver stable performance without requiring additional feature engineering and are widely applied in image classification [2], speech recognition [3], object detection [4], semantic segmentation

[5], face recognition [6], autonomous driving [7], and other domains, attracting extensive attention and research.

In recent years, with the development of mobile internet and breakthroughs in data storage capacity, massive, multimodal, and high-value datasets have emerged [8]. Numerous researchers and companies have attempted to extract valuable information from these datasets. However, the sheer volume of data leads to substantial time consumption for DCNN model training, while data and modality variations require repeated parameter tuning. Consequently, reducing the cost of DCNN model training in big data environments has become an urgent challenge.

Google' s MapReduce parallel computing model, with its advantages of easy programming, high fault tolerance, load balancing, and strong scalability, has gained favor among scholars and enterprises. Many DCNN algorithms based on the MapReduce model have been extensively studied [9–12]. Reference [13] proposes a MapReduce-based parallel DCNN algorithm that adopts a divide-and-conquer approach, using MapReduce' s Split method to partition data, constructing multiple computing nodes to simultaneously train DCNN models, and selecting the network model with the highest accuracy as output, achieving parallelized DCNN training. Building upon this, reference [14] proposes the FCNN algorithm (Fully CNN for processing CT scan images), which transforms full views into sparse views and applies Gaussian filters to smooth feature edges, enhancing important texture information. Although converting full views to sparse views accelerates reading speed, the altered feature structure of sparse views makes feature selection difficult, resulting in excessive data redundancy during model training. Reference [15] proposes the SSOCNN algorithm (An optimization of im2col, an important method of CNNs based on continuous address access) based on the Im2col method. This algorithm designs an Im2col acceleration method for single-stride scenarios using continuous memory address access, accelerating the process of mapping images to matrices by changing data reading order and utilizing general matrix multiplication for column vector and convolution kernel operations, thereby accelerating convolutional layer computation. This is essentially a model parallelism approach. However, during parallel convolution construction, the algorithm struggles to eliminate redundant convolution kernels distributed across nodes, failing to address slow convolutional layer operations in big data environments. Reference [16] combines DCNN with firefly algorithms to propose MR-FPDCNN (Deep convolutional neural network algorithm based on feature graph and parallel computing entropy using MapReduce). This algorithm integrates information-sharing search strategies with firefly algorithms to find optimal network parameters and shares DCNN network parameters through MapReduce communication mechanisms to accelerate loss function convergence. This is essentially a data parallelism approach. However, the algorithm does not identify and handle abnormal node training data, causing loss function convergence oscillation during backpropagation and resulting in poor convergence.

In summary, although the aforementioned algorithms have achieved certain results, problems such as excessive data redundancy, slow convolutional layer operations, and poor loss function convergence remain urgent challenges. To address these issues, this paper proposes a parallel deep convolutional neural network optimization algorithm based on the Im2col method (IA-PDCNNOA) on the MapReduce parallel computing framework. The algorithm optimizes parallel deep convolutional neural networks in three aspects: (a) In the feature parallel extraction stage, existing algorithms mostly perform conventional preprocessing such as noise removal, missing value handling, data standardization, and normalization. However, in big data environments, these conventional operations not only fail to improve model accuracy but also cannot reduce computational resource consumption. The IA-PDCNNOA algorithm improves the Marr-Hildreth operator to propose the MHO-PFES strategy, which extracts all edge features from image data, evaluates feature similarity among similar data types, and removes low-similarity features to solve excessive data redundancy. This eliminates the impact of irrelevant features on model training accuracy while reducing computational overhead. (b) In the model parallel training stage, the algorithm proposes the IM-PMTS strategy based on data parallelism. It designs Mahalanobis distance center values to remove redundant convolution kernels in the current layer and combines Im2col with MapReduce for parallel training to accelerate convolution operations, thereby improving convolutional layer operation speed. (c) In the parameter parallel update stage, most parallel algorithms use stochastic gradient descent or batch gradient descent for parameter updates during backpropagation. However, they fail to consider that abnormal nodes caused by program interruptions, memory overflow, and other factors can lead to abnormal training data, which causes loss function convergence oscillation and poor convergence. The IA-PDCNNOA algorithm proposes the IM-BGDS strategy based on model parallelism, which evaluates the distance between each node' s gradient and the batch gradient to remove outliers from the batch gradient, enabling adaptive batch gradient adjustment and solving the poor convergence problem.

In summary, the main contributions of this paper include: (a) Proposing the MHO-PFES strategy to extract target features from data as convolutional neural network input, solving the problem of excessive data redundancy; (b) Proposing the IM-PMTS strategy, which removes redundant convolution kernels by designing Mahalanobis distance center values and combines MapReduce with Im2col for parallel model training, improving convolutional layer operation speed; (c) Proposing the IM-BGDS strategy to eliminate the influence of abnormal node training data on batch gradients, solving the problem of poor loss function convergence.

1 Related Concepts

Definition 1 (Non-local Means Algorithm [17]). The non-local means algorithm is a neighborhood pixel-based filtering method that combines global image information for data denoising. Let x represent a data sample and y represent a neighborhood window. The denoised grayscale value can be expressed as $\omega(x,y) = \sum_{y\in\psi(x)} \phi(x,y)\cdot\theta(y)$, where $\psi(x)$ denotes the search window, $\phi(x,y)$ represents the similarity between regions x and y, and $\theta(y)$ is the noisy image.

Definition 2 (Cosine Similarity [18]). Cosine similarity measures similarity between individuals by mapping their indicator data to vector space and measuring the cosine of the angle between two individual vectors. Let X and Y represent comparison individuals, with x and y denoting their one-dimensional forms. Cosine similarity $Sim(X,Y)$ can be expressed as $Sim(X,Y) = \cos\theta = \frac{x\cdot y}{|x|+|y|}$, where $|x|$ and $|y|$ are the magnitudes of the individuals.

Definition 3 (Image to Column, Im2col [19]). Im2col is a transformation function that converts convolution computation into matrix multiplication. It transforms the input' s 3D matrix into a 2D matrix and convolution kernels into 1D matrices, enabling convolution computation through matrix multiplication. Let I represent the input feature map, K represent the convolution kernel, and (x,y) denote pixel coordinates. The convolution result O can be expressed as $O_{x,y} = \sum_{i=0}^{k_H}\sum_{j=0}^{k_W} I_{x+i,y+j}\cdot K_{i,j}$, where k_H and k_W are the height and width of the kernel set.

Definition 4 (Mahalanobis Distance [20]). Mahalanobis distance is a common distance metric in metric learning and an effective method for calculating similarity between two unknown sample sets. Let S represent the covariance matrix of multidimensional random variables, μ represent the sample mean, and x represent the current data point. Mahalanobis distance $MD(x)$ can be expressed as $MD(x) = \sqrt{(x-\mu)^T S^{-1}(x-\mu)}$.

The IA-PDCNNOA algorithm consists of three stages: feature parallel extraction, model parallel training, and parameter parallel update. (a) Feature parallel extraction stage: The MHO-PFES strategy is proposed to first extract data features from the original dataset using an improved Marr-Hildreth operator, then filter target features as input to the convolutional neural network, solving the problem of excessive data redundancy. (b) Model parallel training stage: The IM-PMTS strategy is proposed to first design Mahalanobis distance center values for pruning convolution kernels in the same layer, then accelerate the convolution operation process through parallel training combining MapReduce and Im2col methods, improving convolutional layer operation speed. (c) Parameter parallel update stage: The IM-BGDS strategy is proposed to first construct loss sum gradients for building mini-batch data gradients, then update parameters in parallel through backpropagation, eliminating the influence of abnormal node training data on batch gradients and solving the poor convergence problem.

2.1 Feature Parallel Extraction

Current parallel DCNN algorithms in big data environments contain numerous redundant features in initial image data that are not effectively filtered, causing excessive data redundancy during model training. To address this, we propose the MHO-PFES strategy based on the Marr-Hildreth operator, which consists of two steps: (a) Feature extraction: An improved non-local mean filter $(FT)_{a,b}$ filters input image data, and the Laplacian equation of the filtered data is computed to extract image features by finding zero crossings of the Laplacian equation. (b) Feature screening: To further filter target features, we propose the Feature Correlation Index $(FCI)_{x,y}$ to compare similarity between any two image blocks, setting a correlation coefficient ε and removing image blocks where $(FCI)_{x,y} < \varepsilon$ to reduce redundant features in the data.

1) Feature Extraction

To obtain high-precision image features, the initial dataset must first be denoised. Therefore, we propose a cosine similarity-based non-local mean filter $(FT)_{a,b}$ that removes data noise through image self-similarity across different regions. The specific process is: First, set a neighborhood window matrix centered at pixel point a and a search window matrix centered at pixel point b on the target image, sliding the neighborhood window across the image. Obtain weighted values for the neighborhood window by comparing cosine similarity of pixel point b' s matrix, then perform denoising based on weight values and each point' s grayscale value to obtain denoised image $g(x, y)$. Next, set a 3$×$3 convolution kernel $h_{a,b}$ to obtain the Laplacian equation $\nabla^2 g(x, y) = g(x, y) * h_{a,b}$. Finally, determine whether the current node' s second derivative of the Laplacian equation is a zero crossing and whether its first derivative is at a large peak. If both conditions are satisfied, retain the node; otherwise, set the pixel to zero. Merge the current data nodes to obtain feature-extracted data $\rho(x, y)$.

Theorem 1 (Cosine Similarity-Based Non-Local Mean Filter $(FT)_{a,b}$). Let $\omega_{x,y}$ and $\psi_{x,y}$ represent the neighborhood window matrix centered at pixel point x and the search window matrix centered at pixel point y, respectively. The transformation function calculation formula is $\rho_{x,y} = \omega_{x,y} + \psi_{x,y}$, where $\omega_{x,y} = \sum_{i=1}^{k} \theta_i \cdot G_{a,b}$ and $\theta_i = \frac{1}{k} \sum_{i=1}^{k} \rho_{x,y}^{i}$.

Proof. The non-local mean filtering principle utilizes the non-correlation characteristic of noise. Let the noise-free pixel block value be $\omega_{x,y}$, the noise value be $\psi_{x,y}$, and the noise-fused pixel block value be $\rho_{x,y}$. The expectation after stacking similar pixel blocks and averaging is $E[\rho_{x,y}] = \frac{1}{k} \sum_{i=1}^{k} (\omega_{x,y}^{i} + \psi_{x,y}^{i})$. Due to pixel block similarity, $E[\omega_{x,y}] = \omega_{x,y}$ (noise-free) with variance σ_{ω}^{2}. When noise is 0, $E[\psi_{x,y}] = 0$. Additionally, due to noise non-correlation, $Var[\rho_{x,y}] = \frac{1}{k^2} \sum_{i=1}^{k} (\sigma_{\omega}^{2} + \sigma_{\psi}^{2}) = \frac{\sigma_{\psi}^{2}}{k}$. This shows noise variance is related to k, and $(FT)_{a,b}$ reduces data noise.

2) Feature Screening

After feature extraction, the strategy partitions the data into image blocks and proposes the Feature Correlation Index $(FCI)_{x,y}$ to calculate feature similarity between any two image blocks, removing image blocks where $(FCI)_{x,y} < \varepsilon$ to eliminate redundant data. The specific process is: First, split images of the same category into equal-sized blocks and propose $(FCI)_{x,y}$ to calculate similarity between any two blocks, where x and y represent two distinct image blocks. Next, map and store $(FCI)_{x,y}$ to HDFS, set a correlation coefficient ε, and remove key-value pairs where $(FCI)_{x,y} < \varepsilon$ to reduce redundant features. Finally, traverse the key-value pairs again, read remaining block IDs from HDFS, and use these filtered blocks as convolutional neural network input to complete feature screening.

Theorem 2 (Feature Correlation Index $(FCI)_{x,y}$). Let x and y represent two feature vectors, μ_x and μ_y represent their expectations, and σ_x^2 and σ_y^2 represent their variances. The Feature Correlation Index is calculated as $(FCI)_{x,y} = \frac{\sigma_{xy}^2}{\sigma_x^2+\sigma_y^2+(\mu_x-\mu_y)^2}$, where σ_{xy}^2 is the covariance between x and y.

Proof. $(FCI)_{x,y}$ measures feature similarity between x and y. Let μ_x and μ_y be expectations, σ_x^2 and σ_y^2 be variances. When feature vectors x and y are linearly related, convolution operations on them involve linear superposition that cannot extract features, resulting in $(FCI)_{x,y} \to 0$. When x and y have similar features, $\sigma_{xy}^2 \to \sigma_x^2 = \sigma_y^2$ and $\mu_x = \mu_y$, making $(FCI)_{x,y} \to 1$.

Algorithm 1 Feature Parallel Extraction Algorithm

Input: Batch data ε_{batch}
Output: Feature-extracted data ε'_{batch}

a) RunMapReduce(ε_{batch})

b) For each chunk i in ε_{batch}
  c) MapReduce.Map($chunk_i$)
  d) For each data block j in $chunk_i$
  e) Apply filter $(FT)_{a,b}$ to block j
  f) Compute Laplacian $\nabla^2 g(x,y)$ and extract features
  g) End Map
  h) MapReduce.Reduce($key, value$)
  i) Split $value$ into blocks
  j) For each block pair (x,y)
  k) Calculate $(FCI)_{x,y}$
  l) Save key-value pair $\langle (FCI)_{x,y}, block \rangle$
  m) End For
  n) While $(FCI)_{x,y} < \varepsilon$
  o) Delete data block where index = x && y
  p) End While
  q) End Reduce

c) End For

d) Return ε'_{batch}

2.2 Model Parallel Training

In current DCNN algorithms for big data environments, parallel model training requires distributing feature maps and convolution kernels to different computing nodes. However, during parallel convolution construction, algorithms struggle to eliminate redundant convolution kernels distributed across nodes, resulting in unresolved slow convolutional layer operations. To address this, we propose the IM-PMTS strategy, which includes: (a) Convolution kernel pruning: Design Mahalanobis Distance Center Value ($MDCV$) to find vectors linearly related to convolution kernels in the network model, compute distances $dist$ from this vector to each kernel, and prune kernels where $dist < \alpha$ to reduce redundant parameters. (b) Parallel Im2col convolution: Use Im2col to map feature graphs into matrices, store matrix-convolution kernel pairs as key-value pairs, distribute them to computing nodes for matrix operations to accelerate convolution, obtain convolution layer results, and store them in HDFS.

1) Convolution Kernel Pruning

To reduce invalid computations from redundant convolution kernels, we design ($MDCV$) to eliminate redundant kernels in the current convolutional layer and accelerate operations. The process is: First, each node computes the covariance matrix S and mean μ of all convolution kernels $\{X_1, X_2, ..., X_n\}$ in the layer, building the objective function $f(x)$. Next, compute the second-order Taylor expansion $f(x_k)+\nabla f(x_k)^T(x-x_k)+\frac{1}{2}(x-x_k)^T\nabla^2 f(x_k)(x-x_k)$ at its stationary point x_k. If the second derivative is non-singular, the next iteration point is $x_{k+1} = x_k - \nabla^2 f(x_k)^{-1}\nabla f(x_k)$. If singular, first determine the search direction $d_k = -\nabla f(x_k)$, then determine the next iteration point $x_{k+1} = x_k + d_k$. Finally, compute distances $dist$ from all convolution kernels to the ($MDCV$) value, set threshold α, and prune kernels where $dist < \alpha$ until optimal kernels are found.

Theorem 3 (Mahalanobis Distance Center Value ($MDCV$)**).** Let $\{X_1, X_2, ..., X_n\}$ be convolution kernels in the model, S be the covariance matrix of all kernels, and μ be their mean. The ($MDCV$) formula is: $(MDCV) = \min_{x\in\mathbb{R}^n} \sqrt{(x-\mu)^T S^{-1}(x-\mu)}$.

Proof. ($MDCV$) measures the distance from feature vector x to feature vector group $\{X_1, X_2, ..., X_n\}$. Let S be the group's covariance matrix and μ the group mean. Introducing S eliminates interference from variable correlations. When feature vector x is linearly related to the group, x is more easily replaced by the group. When $(MDCV) \to 0$, it indicates x is linearly related to $\{X_1, X_2, ..., X_n\}$. Thus, ($MDCV$) represents the minimum distance from x to the feature vector group.

2) Parallel Im2col Convolution

After convolution kernel pruning, we combine MapReduce with Im2col for parallel convolution operations. The process is: First, map input feature map I to convolution computation matrix M using Im2col, and store each mapped matrix M_i with its corresponding kernel as key-value pair $\langle K_i, M_i \rangle$. Next, call the Map() function to perform matrix multiplication between matrix M_i and the corresponding kernel' s 1D vector, obtaining intermediate convolution results. Finally, call Reduce() to merge feature maps for the same data, obtaining the final output feature map O.

Algorithm 2 Model Parallel Training Algorithm

Input: Convolution kernels K, input feature maps I, hyperparameter α
Output: Output feature maps O

a) RunMapReduce(K, I)

b) For each batch i in K, I
  c) MapReduce.Map(K_i, I_i)
  d) Compute ($MDCV$) for convolution kernels
  e) While $dist < \alpha$
  f) Prune redundant kernels
  g) End While
  h) Map I_i to matrix M_i using Im2col
  i) Compute $M_i \times K_i$
  j) End Map
  k) MapReduce.Reduce($key, value$)
  l) Merge results by key
  m) Save key-value pair $\langle O_i, result \rangle$
  n) End Reduce

c) End For

d) Return O

2.3 Parameter Parallel Update

In current parallel DCNN algorithms for big data, distributed cluster nodes first perform forward propagation to obtain convolutional layer results, aggregate them at the Master node, then update parameters via backpropagation using stochastic or batch gradient descent. However, abnormal node training data causes loss function convergence oscillation during backpropagation, leading to poor convergence. To address this, we propose the IM-BGDS strategy with two steps: (a) Gradient construction: Propose Loss Average Weight $(LAW)_g$ to eliminate abnormal node influence and design Loss Sum Gradient $(LSG)_T$

to construct batch data average gradients. (b) Parameter parallel update: After obtaining batch gradients, combine MapReduce with backpropagation error transfer formulas to compute errors in parallel and update parameters.

1) Gradient Construction

To eliminate abnormal node influence on batch gradients, we design $(LAW)_g$ to solve convergence problems. The process is: First, compute the mean of batch data loss functions, calculate the difference between each data point' s loss and this mean to obtain $(LAW)_g$, and map results to key-value pairs $\langle (LAW)_g, (LSG)_T \rangle$ stored in HDFS. Next, compute partial derivatives $\frac{\partial J}{\partial \theta}$ of each data point' s loss function w.r.t. current parameters θ, map results to $\langle (LAW)_g, \frac{\partial J}{\partial \theta} \rangle$ stored in HDFS. Finally, traverse key-value pairs using $(LAW)_g$ as index to construct batch data average gradient $(LSG)_T$ for current parameters.

Theorem 4 (Loss Average Weight $(LAW)_g$). Let g_i represent a data point in batch data, LAD_i be its loss function value, and τ be the absolute difference between LAD_i and the mean loss value. $(LAW)_g$ is calculated as: $(LAW)_g = \begin{cases} 1 & \text{if } LAD_i < \tau \\ 0 & \text{if } LAD_i \geq \tau \end{cases}$.

Proof. $(LAW)_g$ is a weight indicator for data g_i' s loss value. Let $batchsize$ be batch size and τ be the threshold for measuring LAD_i. When $LAD_i < \tau$, the loss value is normal, so set $(LAW)_g = 1$ to retain it. When $LAD_i \geq \tau$, the loss value is abnormal, so set $(LAW)_g = 0$ to exclude it.

Theorem 5 (Loss Sum Gradient $(LSG)_T$). Let x_i represent all data in the batch, $batchsize$ be batch size, and $\frac{\partial J}{\partial \theta_i}$ be the gradient of data x_i' s loss w.r.t. parameter θ. $(LSG)_T$ is calculated as: $(LSG)_T = \frac{1}{batchsize} \sum_{i=1}^{batchsize} (LAW)_g \cdot \frac{\partial J}{\partial \theta_i}$.

Proof. $(LSG)_T$ is the average gradient of batch data x_i. Let $\frac{\partial J}{\partial \theta_i}$ be the gradient of data x_i' s loss w.r.t. parameter θ, and $batchsize$ be batch size. When $(LAW)_g = 1$, data x_i' s gradient descends toward the optimal direction. When $(LAW)_g = 0$, data x_i' s gradient $\frac{\partial J}{\partial \theta_i}$ deviates significantly from the optimal direction and is excluded from the gradient.

2) Parameter Parallel Update

After obtaining batch average gradients, use backpropagation to update error term parameters in parallel, obtaining the updated network model. The process is: First, compute the gradient of layer $l-1$ convolution kernel W_{l-1} w.r.t. loss: $\frac{\partial (LSG)_T}{\partial W_{l-1}} = \sum_{k=1}^{1} \frac{\partial (LSG)_T}{\partial z_k} \cdot \frac{\partial z_k}{\partial W_{l-1}}$, and map results to key-value pair $\langle W_{l-1}, \frac{\partial (LSG)_T}{\partial W_{l-1}} \rangle$ stored in HDFS. Next, compute the parameter change ΔW_{l-1} for convolution kernel W_{l-1} in the network model to update layer $l-1$ parameters. Finally, synchronize updated parameters to all computing nodes via HDFS

for the next update iteration until all network parameters are updated.

Algorithm 3 Parameter Parallel Update Algorithm

Input: Batch data *batch*, model parameters W, τ
Output: Updated network model parameters W'

a) RunMapReduce(*batch*)

b) For each data i in *batch*
   c) Compute $(LAW)_g$ using LAD_i and τ
   d) MapReduce.Map$(i, (LAW)_g)$
   e) Compute gradient $\frac{\partial J}{\partial \theta_i}$
   f) Emit $\langle (LAW)_g, \frac{\partial J}{\partial \theta_i} \rangle$

c) End For

d) For Each key k
   i) MapReduce.Reduce(k, list$\langle \frac{\partial J}{\partial \theta} \rangle$)
   j) Compute $(LSG)_T$ using $\frac{1}{batchsize} \sum (LAW)_g \cdot \frac{\partial J}{\partial \theta}$
   k) Update parameters: $W \leftarrow W - \eta \cdot (LSG)_T$
   l) End Reduce

e) End For

f) Return W'

2.4 IA-PDCNNOA Algorithm Parallelization Flow

The IA-PDCNNOA algorithm' s parallelization flow consists of the following steps:

a) **Feature Parallel Extraction Stage**: Input the original dataset and launch a MapReduce task, partitioning data by category into multiple chunks. Execute the MHO-PFES strategy in mapper nodes to compress the original dataset based on target features and send to reducer nodes, finally storing target features in HDFS.

b) **Model Parallel Training Stage**: Read data from HDFS, randomly shuffle and partition into batches. Launch a new MapReduce task, inputting batch data to mapper nodes to execute the IM-PMTS strategy, performing convolution, ReLU, pooling, etc., to obtain feature maps for the next stage, finally storing prediction outputs in HDFS.

c) **Parameter Parallel Update Stage**: The Master node reads previous stage predictions, executes the IM-BGDS strategy, computes batch gradients for fully connected and convolutional layer parameters via backpropa-

gation. After multiple iterations of steps (b) and (c), find the loss function minimum to obtain the final trained network model.

The IA-PDCNNOA algorithm parallelization flow is shown in Figure 1 [Figure 1: see original paper].

2.5 Algorithm Time Complexity Analysis

The IA-PDCNNOA algorithm' s time complexity comprises three steps: feature parallel extraction, model parallel training, and parameter parallel update.

a) **Feature Parallel Extraction Stage**: This combines MapReduce with parallel unit operations. Its time complexity includes: (i) Data feature extraction under parallel architecture; (ii) Master node feature screening. Let sample count be n, cluster node count be m, and maximum Laplacian iteration count be k. The feature extraction stage uses two sliding windows to traverse data and compute zero crossings, with complexity $O(m \cdot n \cdot k \cdot \log n)$. The feature screening stage computes similarity between any two data slices with complexity $O(n^2)$. Thus, feature parallel extraction complexity is $T_1 = O(m \cdot n \cdot k \cdot \log n) + O(n^2)$.

b) **Model Parallel Training Stage**: This includes (i) $(MDCV)$ value solving; (ii) Parallel convolution operations. Let node count be p, convolution kernel count be s, kernel size be k, and sample count be n. The $(MDCV)$ solving process requires computing kernel standard deviation and finding maximum linear correlation, with complexity $O(p \cdot s \cdot k)$. After previous data processing, convolution operations only require matrix multiplication, with complexity $O(n^2)$. Thus, model parallel training complexity is $T_2 = O(p \cdot s \cdot k) + O(n^2)$.

c) **Parameter Parallel Update Stage**: This involves batch gradient construction. Let node count be c and fully connected input size be k. Batch gradient construction complexity is $O(c \cdot k^2)$. Thus, parameter parallel update complexity is $T_3 = O(c \cdot k^2)$.

Overall, IA-PDCNNOA algorithm time complexity is $T = T_1 + T_2 + T_3 = O(m \cdot n \cdot k \cdot \log n) + O(p \cdot s \cdot k) + O(c \cdot k^2) + O(n^2)$.

For the FCNN [14] algorithm, which first converts sparse views to full views then builds data reconstruction and processing techniques for high-precision parallel training, its time complexity is $T_{FCNN} = O(n \cdot n \cdot s \cdot d \cdot \log n)$, where s is kernel size and d is iteration count.

For the SSOCNN [15] algorithm, which designs continuous memory address reading for single-stride Im2col acceleration and uses general matrix multiplication for convolution, its time complexity is $T_{SSOCNN} = O(a \cdot n \cdot \log n \cdot k^2)$, where a is single-stride count.

For the MR-FPDCNN [16] algorithm, which combines information-sharing search with firefly algorithms to find optimal parameters and trains via MapReduce, its time complexity is $T_{MR-FPDCNN} = O(n \cdot n \cdot p \cdot \log n + k)$, where p is feature map pruning count.

From theoretical analysis, in big data environments where n is significantly larger than other metrics, we have $O(m{\cdot}n{\cdot}k{\cdot}\log n){+}O(p{\cdot}s{\cdot}k){+}O(c{\cdot}k^2){+}O(n^2) < O(n \cdot n \cdot s \cdot d \cdot \log n) < O(a \cdot n \cdot \log n \cdot k^2) < O(n \cdot n \cdot p \cdot \log n + k)$. Thus, IA-PDCNNOA has more favorable time complexity than FCNN, SSOCNN, and MR-FPDCNN in big data environments.

3.1 Experimental Environment

To verify IA-PDCNNOA's performance, we designed relevant experiments. The hardware includes one Master node and seven Slaver nodes, all with AMD Ryzen 7 3800X CPUs, 32GB RAM, and NVIDIA RTX2080Ti GPUs, connected via 1000Mb/s Ethernet.

The programming environment is Python 3.8, TensorFlow 2.3, JDK 1.8, Apache Hadoop 3.3, and Windows 10 Enterprise 2016 LTSB. Node configurations are shown in Table 1 .

Table 1 Configuration of nodes in the experiment

Node Type	Node Name	IP Configuration
Master	master	192.168.111.1
Slave	slave_1~7	192.168.111.2~8

3.2 Experimental Data

Experiments use CIFAR10, CIFAR100, ImageNet 1K, and CompCars datasets:
- **CIFAR10**: 10 classes of $32\$\times32 colorimages, 6,000 imagesperclass, with 50,000 trainingand 10,000 testsample$
$**CIFAR100**: 100 classesof 32\times32 colorimages, 600 imagesperclass, with 500 trainingand 100 testsamplesper$
$**ImageNet1K**: 1,000 classes fromtheworld's largestimagerecognitiondatabase, with 1.2M+$
$trainingand 50,000 validationsamples, resizedto 224\times\$224$ via boundary padding.
- **CompCars**: 208,826 vehicle images across 1,716 models from 163 brands.

Dataset details are shown in Table 2 .

Table 2 Experimental dataset

Dataset	Size/Pixels	Classes	Training Set	Test Set
CIFAR10	32$×32\|10\|50,000\|\|10,000\|\|$*CIFAR100*$\|32×32\|100\|50,000\|\|10,000\|\|$*ImageNet1K*$\|224×224\|1,$			
	$\|50,000\|\|$*CompCars*$|224×$$224			

3.3 Experimental Preparation

We adopt ResNet50 as the training network. As a representative neural network with skip connections, ResNet50 effectively reflects convolutional neural network optimization. To reduce overhead from frequent small file reads, images are converted to grayscale and parallelized via MapReduce to TFRecord format.

Experiments evaluate algorithm performance using four metrics: speedup ratio, accuracy, floating-point operations (FLOPs), and algorithm runtime. Speedup ratio and Top-1 accuracy are defined as follows:

Speedup Ratio: Numerical representation of performance improvement from parallel computation, where higher values indicate greater parallelism: $S = \frac{T_s}{T_p}$, where T_s is serial runtime and T_p is parallel runtime.

Top-1 Accuracy: A critical metric for evaluating prediction error in deep learning, where higher values indicate better performance: $ACC_{Top-1} = \frac{N_{correct}}{N_{total}}$, where $N_{correct}$ is the number of correctly labeled samples in the validation set and N_{total} is the total sample count.

3.5 Algorithm Feasibility Comparison Analysis

To verify IA-PDCNNOA' s parallel training feasibility in big data environments, we measure speedup ratio across CIFAR10, CIFAR100, ImageNet 1K, and CompCars datasets. For accuracy, we average 10 runs per algorithm. Results are shown in Figure 2 [Figure 2: see original paper].

Figure 2 shows that IA-PDCNNOA' s speedup ratio generally increases with node count, growing progressively with dataset scale. At 2 nodes, speedup ratios across the four datasets show minor differences. At 4 nodes, speedup ratios increase by 2.205, 2.417, 3.824, and 4.735 compared to single-node execution. At 8 nodes, IA-PDCNNOA achieves significant improvements of 6.861, 6.876, 9.828, and 8.915 respectively. This is because IM-PMTS distributes equal-sized image blocks evenly across cluster nodes, reducing communication time while ensuring load balance and greatly improving efficiency. Additionally, IM-BGDS excludes abnormal node computations, avoiding resource consumption from reading/writing and transmitting such data, thereby enhancing performance. As data scale increases, this improvement becomes more pronounced,

demonstrating IA-PDCNNOA' s suitability for parallel DCNN model training in big data environments.

3.6.1 Algorithm Speedup Ratio Experimental Analysis

To verify IA-PDCNNOA's parallelization performance in big data environments, we compare speedup ratios against MR-FPDCNN, SSOCNN, and FCNN on CIFAR10, CIFAR100, ImageNet 1K, and CompCars datasets, averaging 10 runs for accuracy. Results are shown in Figure 3 [Figure 3: see original paper].

For smaller datasets (CIFAR10, CIFAR100), all algorithms show gradual speedup ratio increases with node count. At 4 nodes, IA-PDCNNOA' s speedup ratio is 0.325 and 0.435 lower than FCNN, and 0.276 and 0.102 lower than SSOCNN. However, for larger datasets (ImageNet 1K, CompCars), IA-PDCNNOA' s speedup ratio grows substantially, reaching 9.804 and 8.912 at 8 nodes—1.148, 4.173, 4.652 and 0.965, 2.678, 2.094 higher than MR-FPDCNN, FCNN, and SSOCNN respectively. This occurs because, for small datasets, data distribution across nodes incurs rapidly growing communication overhead, limiting speedup gains. For large datasets, IM-PMTS' s ($MDCV$) prunes redundant kernels, reducing network communication overhead, while MapReduce-Im2col parallel training accelerates convolution operations, improving speedup ratios. Experiments show IA-PDCNNOA' s parallelization capability strengthens with cluster node count and dataset scale, making it suitable for large-scale parallel processing.

3.6.2 Algorithm Accuracy Experimental Analysis

To further verify IA-PDCNNOA' s training effectiveness, we evaluate Top-1 accuracy across the four datasets for IA-PDCNNOA, MR-FPDCNN, SSOCNN, and FCNN. Results are shown in Figure 4 [Figure 4: see original paper].

For smaller datasets (CIFAR10, CIFAR100), all algorithms achieve stable high accuracy, with IA-PDCNNOA reaching the highest Top-1 accuracy of 89.72% and 72.31%, converging earlier than MR-FPDCNN, SSOCNN, and FCNN by 2.87%, 4.62%, 6.48% and 2.14%, 4.57%, 3.53% respectively. For larger datasets (ImageNet 1K, CompCars), significant differences in Top-1 accuracy and convergence emerge. IA-PDCNNOA achieves the highest accuracy at 72.41% and 69.17%, exceeding MR-FPDCNN, SSOCNN, and FCNN by 2.31%, 7.98%, 2.85% and 2.81%, 7.58%, 4.84% respectively, while the other three algorithms show varying degrees of convergence difficulty. This is because IA-PDCNNOA' s IM-BGDS strategy designs $(LSG)_T$ to construct mini-batch gradients and updates parameters via backpropagation, excluding abnormal node data from batch gradients and enhancing convergence. Thus, IA-PDCNNOA demonstrates superior

convergence speed and accuracy, making it suitable for large-scale DCNN model parallel training.

3.6.3 Algorithm Runtime and FLOPs Experimental Analysis

To verify IA-PDCNNOA' s execution speed and model optimization in big data environments, we measure runtime and FLOPs for Baseline, IA-PDCNNOA, MR-FPDCNN, SSOCNN, and FCNN across all datasets. Baseline uses ResNet50 with 1/8 data load. Results are shown in Table 3 .

Table 3 Running time and FLOPs of each algorithm on four datasets

Dataset	Algorithm	Running Time (s)	FLOPs
CIFAR10	Baseline	-	3.8$×10^{9}$
	MR-FPDCNN	1.97$×10^{9}$	2.58$×10^{9}$
	SSOCNN	2.39$×10^{9}$	1.78$×10^{9}$
	FCNN	3.8$×10^{9}$	2.05$×10^{9}$
	IA-PDCNNOA	2.87$×10^{9}$	2.94$×10^{9}$
CIFAR100	Baseline	-	3.8$×10^{9}$
	MR-FPDCNN	2.05$×10^{9}$	2.87$×10^{9}$
	SSOCNN	2.94$×10^{9}$	1.91$×10^{9}$
	FCNN	3.8$×10^{9}$	2.62$×10^{9}$
	IA-PDCNNOA	6.23$×10^{4}$	3.09$×10^{9}$
ImageNet	Baseline	8.12$×10^{4}$	3.8$×10^{9}$
1K	MR-FPDCNN	6.23$×10^{4}$	2.62$×10^{9}$
	SSOCNN	8.76$×10^{4}$	3.09$×10^{9}$
	FCNN	1.02$×10^{5}$	2.81$×10^{9}$
	IA-PDCNNOA	4.91$×10^{4}$	2.5$×10^{9}$
CompCars	Baseline	6.72$×10^{4}$	3.8$×10^{9}$
	MR-FPDCNN	4.64$×10^{4}$	2.49$×10^{9}$
	SSOCNN	7.72$×10^{4}$	2.98$×10^{9}$
	FCNN	9.18$×10^{4}$	2.96$×10^{9}$
	IA-PDCNNOA	3.59$×10^{4}$	2.41$×10^{9}$

For smaller datasets (CIFAR10, CIFAR100), runtime differences are minor but FLOPs reduce significantly: IA-PDCNNOA reduces FLOPs by 5%, 21%, 16% and 5%, 25%, 27% compared to MR-FPDCNN, SSOCNN, and FCNN respectively. For larger datasets (ImageNet 1K, CompCars), IA-PDCNNOA outperforms others in both runtime and FLOPs: runtime is faster by $1.32\$\times10^{\{4\}s}, 3.85\times10\{4\}s, 5.29\times10^{\{4\}sand}1.05\times10\{4\}s, 4.13\times10^{\{4\}s}, 5.59\times10\{4\}\$s$ respectively, with FLOPs reduced by 3%, 13%, 8% and 3%, 15%, 15%. As dataset size increases, IA-PDCNNOA' s runtime and FLOPs reduction widens

the gap with other algorithms. This is because MHO-PFES' s $(FCI)_{x,y}$ removes redundant features and filters target features, reducing FLOPs and accelerating execution. Thus, IA-PDCNNOA outperforms MR-FPDCNN, SSOCNN, and FCNN for large-scale DCNN parallel training.

3.6.4 Algorithm Parallel Mode Performance Experimental Analysis

To verify parallel mode impact on model construction time in big data environments, we compare IA-PDCNNOA (data-parallel forward, model-parallel backward) against data-parallel MR-FPDCNN and model-parallel SSOCNN. We measure runtime to reach 70% accuracy on all datasets. Results are shown in Figure 5 [Figure 5: see original paper].

For small datasets (CIFAR10, CIFAR100), training times are similar. However, for large datasets (ImageNet 1K, CompCars), IA-PDCNNOA reduces runtime by 1322s and 3837s compared to MR-FPDCNN, and 1049s and 4127s compared to SSOCNN. As dataset scale increases, IA-PDCNNOA' s training time advantage becomes significant. This is because data-parallel MR-FPDCNN distributes data to separate nodes without parameter sharing, requiring longer training times. Model-parallel SSOCNN incurs heavy communication overhead from merging feature maps across nodes. IA-PDCNNOA' s hybrid approach eliminates inter-node communication during forward propagation while building batch gradients during backpropagation, substantially reducing runtime. Experiments show IA-PDCNNOA' s hybrid parallelism is more suitable for large-scale DCNN training.

---

4 Conclusion

To address traditional DCNN limitations in big data environments, this paper proposes IA-PDCNNOA, a parallel deep convolutional neural network optimization algorithm based on Im2col. First, MHO-PFES filters input data using an improved non-local mean filter $(FT)_{a,b}$, computes the Laplacian equation $\nabla^2 g(x,y)$, extracts features via zero-crossings, and proposes $(FCI)_{x,y}$ to remove redundant data, solving excessive redundancy. Second, IM-PMTS designs Mahalanobis distance center value $(MDCV)$ to find vectors linearly related to convolution kernels for same-layer pruning, then combines MapReduce and Im2col for parallel training to accelerate convolution and improve operation speed. Finally, IM-BGDS designs loss average weight $(LAW)_g$ to exclude abnormal node influence and loss sum gradient $(LSG)_T$ to construct batch average gradients, combining MapReduce with backpropagation for parallel parameter updates that solve poor convergence.

To validate IA-PDCNNOA' s performance, we design experiments on ResNet50 across CIFAR10, CIFAR100, ImageNet 1K, and CompCars datasets, comparing against MR-FPDCNN, SSOCNN, and FCNN. Results and analysis demonstrate IA-PDCNNOA' s superior performance for large-scale DCNN parallel training. While IA-PDCNNOA advances DCNN model training, prediction accuracy and parallel performance still have room for improvement, representing future research focus.

References

[1] Zhang Ke, Feng Xiaohan, Guo Yurong, et al. A review of deep convolution neural network models for image classification [J]. Journal of Image and Graphics, 2021, 26(10): 21.

[2] Yang Zhenzhen, Kuang Nan, Fan Lu, et al. Review of Image Classification Algorithms Based on Convolutional Neural Networks [J]. Journal of Signal Processing, 2018, 34(12): 1474-1489.

[3] Zhu Fangyuan, Ma Zhiqiang, Chen Yan, et al. Survey of Speaker Adaptation Methods in Speech Recognition [J]. Journal of Frontiers of Computer Science and Technology, 2021, 15(12): 15.

[4] Luo Huilan, Yuan Pu, Tong Kang. Review of the Methods for Salient Object Detection Based on Deep Learning [J]. Acta Electronica Sinica, 2021, 49(7): 11.

[5] Xu Hui, Zhu Yuhua, Zhentong, et al. Survey of Image Semantic Segmentation Methods Based on Deep Neural Network [J]. Journal of Frontiers of Computer Science and Technology, 2021, 15(1): 13.

[6] Bai Ziyi, Mao Yirong, Wang Ruiping. Survey on Video-based Face Recognition [J]. Computer Science, 2021, 48(3): 10.

[7] Zhu Xianglei, Wang Haichi, Youhammer, et al. Survey on Testing of Intelligent Systems in Autonomous Vehicles [J]. Journal of Software, 2021, 32(7): 22.

[8] Fairuz, Amalina, Ibrahim, et al. Blending Big Data Analytics: Review on Challenges and a Recent Study [J]. IEEE Access, 2020, 8: 3629-3645.

[9] Vitor C. F. Gomes, Gilberto R. Queiroz, Karine R. Ferreira. An Overview of Platforms for Big Earth Observation Data Management and Analysis [J]. Remote Sensing, 2020, 12(8): 1253-1253.

[10] Xiao Wen, Hu Juan, Zhou Xiaofeng. Parallel association rules mining algorithm based on MapReduce: a survey [J]. Application Research of Computers, 2018, 35(01): 13-23.

[11] Jin Guodong, Bian haoqiong, Chen Yueguo, et al. Survey on Storage and

Optimization Techniques of HDFS [J]. Journal of Software, 2020, 31(1): 137-161.

[12] Mahdi Mahmoud A. and Hosny Khalid M. and Elhenawy Ibrahim. Scalable Clustering Algorithms for Big Data: A Review [J]. IEEE ACCESS, 2021, 9: 80015-80027.

[13] Leung J, Chen M. Image Recognition with MapReduce Based Convolutional Neural Networks [C]// 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON). IEEE, 2019, pp. 0119-0125.

[14] Takam C A, Samba O, Kouanou A T, et al. Spark Architecture for deep learning-based dose optimization in medical imaging [J]. Informatics in Medicine Unlocked, 2020, 19: 100335.

[15] Wang H, Ma C. An optimization of im2col, an important method of CNNs, based on continuous address access [C]// 2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE). 2021, pp. 314-320.

[16] Mao Yimin, Zhang ruipeng, Gao Bo. Deep convolutional neural network algorithm based on feature map in big data environment [J/OL]. Computer Engineering and Applications, 2022, 1-9. [2022-02-07]. http://kns.cnki.net/kcms/detail/11.2127.TP.20210413.1143.010.html.

[17] Park J, Kang C K, Lee Y. Quantitative evaluation of the image quality using the fast nonlocal means denoising approach in diffusion-weighted magnetic resonance imaging with high b-value [J]. Journal of the Korean Physical Society, 2021, 78(3): 244-250.

[18] Chen Jun, He Qing. Improved butterfly optimization algorithm based on cosine similarity [J]. Journal of Computer Applications, 2021, 41(09): 2668-2677.

[19] Ji Z, Zhang X, Wei Z, et al. A tile-fusion method for accelerating Winograd convolutions [J]. Neurocomputing, 2021, 460: 9-19.

[20] Wang Yan, Qi Xianghui, Duan Yaxi. Image segmentation of FCM algorithm based on kernel function and Markov distance [J]. Application Research of Computers, 2020, 37(02): 611-614+624.

Note: Figure translations are in progress. See original paper for figures.