EAI Endorsed Transactions
on Internet of Things
Research Article EAI.EU

# Circumventing Stragglers and Staleness in Distributed CNN using LSTM

Aswathy Ravikumar[1,*], Harini Sriraman[2], Saddikuti Lokesh[3] and Jitendra Sai[4]

[1,2,3,4] Vellore Institute of Technology, Chennai, India

Abstract

INTRODUCTION: Using neural networks for these inherently distributed applications is challenging and time-consuming. There is a crucial need for a framework that supports a distributed deep neural network to yield accurate results at an accelerated time.
METHODS: In the proposed framework, any experienced novice user can utilize and execute the neural network models in a distributed manner with the automated hyperparameter tuning feature. In addition, the proposed framework is provided in AWS Sage maker for scaling the distribution and achieving exascale FLOPS. We benchmarked the framework performance by applying it to a medical dataset.
RESULTS: The maximum performance is achieved with a speedup of 6.59 in 5 nodes. The model encourages expert/ novice neural network users to apply neural network models in the distributed platform and get enhanced results with accelerated training time. There has been a lot of research on how to improve the training time of Convolutional Neural Networks (CNNs) using distributed models, with a particular emphasis on automating the hyperparameter tweaking process. The study shows that training times may be decreased across the board by not just manually tweaking hyperparameters, but also by using L2 regularization, a dropout layer, and ConvLSTM for automatic hyperparameter modification.
CONCLUSION: The proposed method improved the training speed for model-parallel setups by 1.4% and increased the speed for parallel data by 2.206%. Data-parallel execution achieved a high accuracy of 93.3825%, whereas model-parallel execution achieved a top accuracy of 89.59%.

**Keywords:** Convolutional Neural Network, AWS Sage Maker, Distributed Framework, Parameter Server, Exa-Scale Computing, Distributed Autotuning

Received on 24 November 2023, accepted on 03 February 2024, published on 14 February 2024



doi: 10.4108/eetiot.5119

*Corresponding author. Email: aswathyravi2290@gmail.com

## 1. Introduction

Similar to the biological neural systems of the brain, Deep Neural Networks must also train from unfamiliar input through hierarchical training. The training utilizes mathematical methods to identify changes in the connections of neurons. The network performs computations and makes predictions based on what it has learned. Therefore, the first layers of the network enhance the understanding of the deeper layers. Training a DNN with massive datasets might take days. Despite the precision and effectiveness of the training, experts are equally concerned with enhancing its performance execution. It is vital to employ parallel and distributed settings to improve the speed of programs that deal with large amounts of data and execute intensive calculations. In addition, DNN models conduct many matrix multiplications that may be performed on a GPU [1], [2]. This kind of system is in great demand, and there is a lot of active study happening in this field because of its potential industrial and other uses. In the realm of image processing, Convolutional Neural Networks (CNNs) have emerged as a significant form of deep learning network. The convolutional neural network (CNN), commonly used in deep learning techniques for image processing, will require significant improvement as data volumes continue to grow. CNNS must increase its efficiency as technology and data continue to progress. To

build a better distributed deep learning model, you need to carefully examine how each CNN layer affects the whole network. To get the best results, neural networks aim to simulate the way the human brain functions by using networks of linked neurons that can learn and adapt on their own. Many different neural networks are available for use in deep learning (DL) software to tackle these sorts of problems.

Improving performance in terms of training time and result accuracy requires understanding previous designs and developing an architecture that is both precise and efficient. CNNs are specialized neural networks that can recognize objects, faces, and perform a wide variety of other tasks. They mostly struggle with issues like how to get a big, well-curated dataset and how to achieve rapid training rates. The reliability of the model may suffer if there is not enough data to train it properly. In general, CNNs are considered among the most complicated neural networks, necessitating precise management and optimization because of their inherent complexities. There are a lot of different approaches to making a neural network more accurate and reducing the amount of time it takes to train it. Two methods for cutting down the amount of time spent training are the drop-out layer and early termination. The dropout method is widely considered to be one of the most successful approaches to optimization. Following the max-pooling layer, a dropout layer is added to further tune the weights of the network. Dropout provides a big helping hand to the model in preventing it from over- or under-fitting the data. According to the models, the value of the dropout probability will fall somewhere in the range of 0 and 0.5 for a wide variety of datasets and models. In most cases, nodes that are prone to dropouts have dropout probability values that are either less than 1 or equal to 1. Recent investigations have shown that CNN and LSTM may be coupled to adjust hyperparameters when used together. The model can attain the best possible degree of precision with the assistance of LSTM and CNN. The challenges that might arise with models' short-term memories can be alleviated by a variation of RNN known as LSTM. Applications in the real world of natural language processing are its principal use.

Several strategies have been offered to speed up the learning process in recent years [3-5]. Initially, they focus on investigating the fluctuation of network characteristics, like loss and activation functions. Subsequent techniques began investigating architectural advances in network design, such as expanding the depth and breadth of models [6]. TensorFlow and Theano [7], ML reference libraries, already offer training distribution across parallel resources in multi-node setups. However, they demand a greater understanding of parallel and distributed models using programming paradigms such as CUDA and MPI and novel distributed computing methods. Distributed Deep Learning frameworks may use large-scale devices to accelerate the training of deep neural networks (DNN) without additional programming effort. These solutions use ML frameworks to spread movement from a single machine to a multi-node network, including several devices. Due to the rapid evolution of the subject, new DDL systems are often introduced, with support for many DL models, parallel computing approaches, and parallelization methodologies. Components of DNNs, like the SGD dataset partition into mini batches, promote parallel training. Minibatches show different input parts processed by employees. Workers must interact at a given time to share data. Hence parameter updates are determined by averaging gradients over all input. This represents a typical situation in parallel computing, in which we divide a problem into numerous tasks, distribute them over multiple workers, and, if required, create synchronization nodes. The key challenges when trying to design DL methodologies, which are the considerable sharing of information between processes, the collection of the dataset and the model on each worker based on the employed strategy, and the performance enhancements to minimize the processing time and keep or enhance model accuracy. DDL techniques are continually emerging. Novel DDL frameworks are now being designed or enhanced so that training may be accelerated without compromising convergent efforts to decrease estimation error. Several learning features must be considered when expanding for additional devices, including the training's objectives, the overall network model employed, and the frameworks' parallel method.

The objectives of the paper are:

i. To develop a model, which any experienced /novice user can utilize and execute the neural network models in a distributed manner in AWS Sage maker for scaling the distribution and achieving exascale FLOPS.

ii. To ensure easy scalability and hyper-tunning of the DDL models in the cloud.

iii. Enhancing the learning curve in a distributed neural network by automating the optimization of hyperparameters remains a persistent challenge.

iv. A significant obstacle that remains unresolved in distributed environments is finding ways to reduce training time while maintaining high accuracy levels.

v. The examination of the consequences of scaling a DNN is an area that has yet to be thoroughly explored, and this research aims to provide techniques for addressing this problem.

## 2. Related Work

In the latest days, many frameworks have been adopted and reviewed owing to the rapid sector innovation rate. The following sections review several accessible DDL frameworks, previous publications that provided a performance evaluation of distributed architectures, and the most popular methods for assessing the effectiveness of distributed learning. Existing frameworks for accelerating deep understanding by using numerous nodes in massive HPC systems are continually being developed and enhanced. The initially distributed training options gave new capabilities to ML-savvy users, who had to acquire new ideas on how to redistribute training layers & stages to workers on various nodes. Recently, new frameworks have been developed to restrict users from gaining a

comprehensive understanding of high-performance systems to enhance their training performance—these frameworks abstract system settings to facilitate the rapid transformation of single-node learning into multi-node learning.

The Uber Engineering group created Horovod for simple Python data parallelism distribution on top of TensorFlow, Keras, or PyTorch. Horovod employs the MPI and pthreads enabling parallel computing on the backend. Tarantella is a new distributed computing tool created by the Competence Center for High-Performance Computing that implements data parallelism. It is open-source and includes comprehensive documentation and developer contact information. Whale is a newly developed framework for training enormous models leveraging data and model parallelism approaches. Finally, Orca is a component of the BigDL 2.0 initiative and offers an intuitive framework for scaling single-node Python training to multiple nodes. The powerful frameworks are listed in Table 1.

Table 1. Existing Distributed Frameworks

| Framework | Features | | |
|---|---|---|---|
| | Language | Subheading | Language |
| Horovod | Python and C++ | Python and C++ | Python and C++ |
| Orca | Python | Python | Python |
| Tarantella | Python and C++ | Python and C++ | Python and C++ |
| Whale | Whale Python | Whale Python | Whale Python |

## 2.1. Convolutional Neural Network

The discipline of machine learning has reaped major benefits in recent years from the implementation of DL-based approaches. In addition to this, its performance in the realm of machine learning frequently outstrips that of other computational methods. Because it also offers the maximum degree of precision for a wide range of complicated models and, in many cases, surpasses the skills of highly evolved humans. The key advantage that comes with using DL models is that they can effectively manage extremely large amounts of data. This is proved by the fact that the sector is widely utilized. DL has surpassed standard ML approaches in a range of sectors, including cyber security, bioinformatics, language processing, robotics, and medical information processing, to name just a few of these areas of study. Using a CNN and an LSTM (long short-term memory)-based DL method, the purpose of this work is to identify corona using x-rays. CNN is used to extract more in-depth features, while LSTM is used to identify those features once they have been derived. Within the 4575 X-ray photos that make up the collection for this system are 1526 shots that depict the corona. They have presented a technique in this research that makes use of the findings of testing to produce an accuracy of 99.4, an area under the curve (AUC) of 99.9, and a specificity of 99.9.

In today's world, it is becoming increasingly important to optimize CNNs in terms of accuracy, the amount of time required for training, and scalability. Understanding the structure of CNN is necessary if one is to get optimal performance using CNN. [1,18] analyzes CNN's structure on a variety of CPU, TPU, and GPU devices for the sake of benchmark applications. This article demonstrates that there is a correlation between the structure of CNN and its performance. Li Wang and Jason Kuen investigated the development of CNN in some different respects, including layer building, activation function, job loss, familiarity, efficiency, and expeditious computation. The writers of this piece also investigated CNN applications for natural language processing (NLP), computer vision, and speech. The approach for assessing biorepositories in a low exposure setting that is suggested in [9] is based on statistical analysis and the neural network separation technique. This method was developed. The findings shed light on the relative number of unique biorepositories present within the sample. Over the past few years, CNN has become increasingly useful in a variety of contexts. The length of time required for training a CNN is a significant issue, particularly when the dataset in question is quite large. CNN has put in a lot of work to cut down on its training time while maintaining its high level of accuracy. In the past, accelerators such as TPUs, FPGAs, GPU clusters, and GPUs themselves have been used to speed up CNN training. It is shown in [9] that an overview of several methods may be used to improve the performance of CNN by utilizing ASIC hardware designs. The results of [11] serve as a baseline for comparing the capabilities of GPUs and TPUs while running CNN. Distributed CNN is one of the ways for scaling CNN, even though there are several hardware-centric methods for speeding up CNN performance. In recent years, there has been a surge in the prevalence of Distributed CNN because of the following causes.

## 2.2 Distributed CNN in Cloud Environment

In the work [7], the authors offer a strategy for speeding up distributed deep learning. For short-term load forecasting, a hybrid model consisting of LSTM and CNN has been developed. Even though a great deal of study strategies have been offered as a means of disseminating CNN, the most significant difficulties still concern finding a way to balance training time and correct findings. Tuning hyperparameters has also gotten more challenging for CNN as a result of the incorporation of new features, which is especially true in a distributed context.

## 2.3 Exa Scale Computing

High-Performance Computing (HPC) primarily refers to combining computing capacity to address significant problems in research, technology, or enterprise. Parallel

computation and supercomputing are synonymous phrases in HPC. The essential concept behind HPC would be that example instance, a single computer requires 100 hours to perform a task, but 100 computers may finish the same work in one hour. A single node inside a supercomputer may not even be more efficient, but it may be when all the capabilities are combined. Due to the increasing price of HPC computers, their use was first restricted to a handful of applications, including basic simulation, architecture, and petroleum & gas. Nevertheless, HPC is now applied in various fields, such as data analysis, social media services, and education-related enterprises and areas. For this kind of application, a potent computer machine is required. Pioneers and academics in the field of HPC have argued that 'Exascale systems' should be introduced at the beginning of the next decade. The framework will provide a thousandfold performance boost over present Petascale systems. By completing ExaFlops' worth of computations in seconds, such a powerful HPC machine would allow the deciphering of several scientific puzzles. Based on Exascale computer system development, it has been expected to consist of various nodes whereby each node is outfitted with standard multicore CPUs and so many cores accelerated GPU processors. The emergence of heterogeneity in HPC systems results in. Increasingly complicated platforms as demand increases. The need for more computer power continues to grow. In supercomputing systems, the greatest obstacle is the enormous energy use during HPC information processing. Exa-Flops-level throughput under these restrictive constraints. Nonetheless, a crucial aspect of the method as we go, applications, development platforms, tools, and designs are co-designed. Moreover, hardware Improvement in electricity consumption restrictions is also required.

## 3. Proposed Method

Over the past few years, the field of deep learning has experienced remarkable growth due to its widespread application in real-time scenarios. This research focuses on investigating the impact of CPUs, GPUs, and TPUs on Convolutional Neural Networks (CNNs).
The GPU, which operates with a high number of Arithmetic Logic Units (ALUs), is commonly used for DNN processing. However, it faces challenges related to data access and memory access due to potential data generation issues caused by a single instruction. To improve CNN models' access to Dynamic Random-Access Memory (DRAM), statistical calculations involving floating-point numbers are employed. By reducing matrix size, parameter count, and border length, significant improvements in the model can be achieved.

The study also explores the impact of employing the maximum pooling layer and adjusting parameters on accuracy and training duration. Notably, the dropout layer's influence on convolutional neural networks, irrespective of single or multiple GPU nodes, is examined, with or without the max-pooling layer. Objectives of the proposed model are:

- Designing a user-friendly model for executing neural network models in a distributed manner using AWS SageMaker, enabling scalable distribution and reaching exascale FLOPS.
- Ensuring easy scalability and hyper-tuning of DDL models on the cloud.
- Deploying autotuning CNN for facilitating rapid learning curves.
- Optimizing distributed training time and accuracy while performing CNN distribution.

Additionally, this article provides a comprehensive study that explores the effects of scaling on the proposed model when operating in a cloud environment. CONVLSTM-D-ACCEL can be used by any experienced /novice user who can utilize and execute the neural network models in a distributed manner in AWS Sage maker for scaling the distribution and achieving exascale FLOPS.

Distributed training is when each processing node has a copy of the network, and a dataset is partitioned. Each node receives a unique batch of the dataset, executes a forward and backward pass, and swaps weight modifications for synchronization before going on to the next collection and epoch. Each thread uses the same model but receives different data. In neural networks, data parallelism uses the same weights but different mini batches on each line. Gradients must be synced or averaged after each run. Gradients are synced during the reverse pass and not communicated during the forward pass. The backward pass must provide the gradient to all nodes, which is the method's biggest drawback.

The technique is summarized as follows: Each employee will use a part of the data for training purposes to duplicate the model across all of our workers. Each trainee collects parameters via parameter servers. Every training worker conducts a training cycle and sends the gradients to all parameter servers, which then update the model's parameters. A PS model consists of the following four phases:

1. The centralized parameter server provides all employees with the model weights.
2. Each worker node trains its local model using its local training examples partition and generates its local gradients. Each slaver then transmits its local gradients to the central PS.
3. After collecting all gradients provided by the worker nodes, the PS will sum all gradients.
4. Once the aggregated gradient has been calculated, the parameter server uses it to modify the model's parameters on this centralized server.

In NLP-based applications, particularly for hate speech detection, LSTM has consistently demonstrated superior accuracy compared to other network architectures. This is

attributed to the unique memory cell property of LSTM that distinguishes it from other models. Interestingly, the benefits of LSTM can also be extended to image identification tasks.

Various approaches exist for incorporating LSTM into CNN architectures. In this research proposal, LSTM is integrated into CNN using model classes. Following the convolutional neural network layer, the output from the flattened layer is passed to the subsequent model class, which comprises the LSTM layer, a dense layer, and an output layer. An overview of the ConvLSTM-D-Accel Data-Parallel model and the ConvLSTM-D-Accel Model-Parallel model is presented in Figures 1 and 2, respectively. Additionally, Algorithm 1 and 2 showcases the proposed algorithm for the DConLSTM architecture.

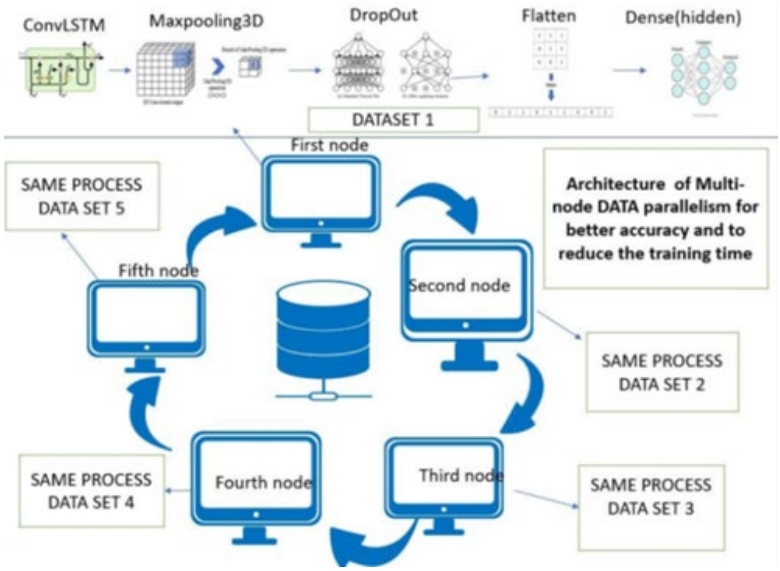


**Figure 1.** Data Parallel Architecture

## 4. Implementation

The dataset used in this example is Pneumonia X-ray images for Pneumonia prediction. The neural network is designed for Pneumonia prediction and is distributed in 5 nodes. The dataset was split among the four nodes (worker nodes), and the central master node acts as the parameter server node. The Neural network was designed using the Convolutional layers and the dense layers to process the 50 X 50 images. The model design is given in Table 2. A single node in AWS Sage Maker for exascale computation sequentially executed the same model. Three benchmark application datasets were selected to be applied to CNN to verify the performance enhancement. They include a dataset for the detection of pneumonia, a dataset for the detection of face masks, a dataset for the detection of diseased leaves, and a dataset for the detection of breast cancer.

Algorithm 1: Pseudo-code Data-Parallel

**Initialization:**

Data will be available in both Master as well as Worker nodes

Data is stored in an array of size m*512

**Initiate** Py4MPI_Threads

**Processing:**

If (rank ==0)

{

**//Master**

Data ={parameter tuning weights contributed by Master}

For I in 1 to no. of nodes

{

Datai = comm.recv(source=I, tag=11)

}

For j in 1 to no. of nodes

{

OptWeight = Func_OptWeight(Data j)

}

Broadcast(OptWeight, allnodes)

}

Else if (rank >0)

{ /* For ‘i’th node */

comm.send (Datai, dest=0, tag=11)

Wait for synchronization

updated weight = OptWeight

For k no. of iterations

{

Process using updated weights

}

}

Algorithm 2: Pseudo-code Model-Parallel
**Initialization:**
Same data available in all nodes
Data is stored in an array of size m*512
**Initiate** Py4MPI_Threads
**Processing:**
If (rank ==0)
{
Wait for synchronization
**Weights**.The model result from Node i
Append. Layers(ith model of Layer i)
}
Else if (rank > 0)
{
for node 'i' in the network
{
ith Model=sequential()
ith Model = add.layer()
Datai = comm.send (dest=0, tag=11)
}
}
if (rank >0)
{
Wait for synchronization.
updated weight = OptWeight
For k no. of iterations
{
Process using updated weights
}
}

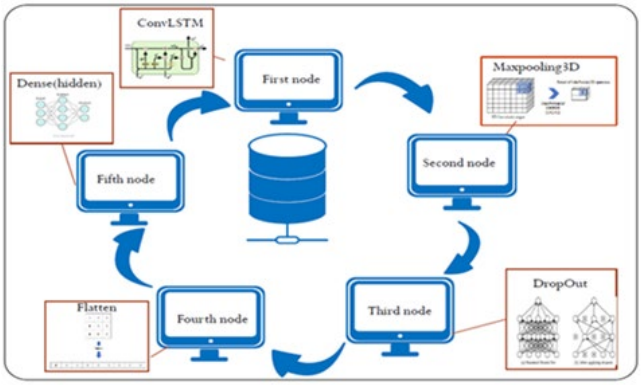


**Figure 2.** Model Parallel Architecture

Table 2. CNN Model Topology

| Parameters | Data |
|---|---|
| Input size | 50x50 |
| Conv2D –ReLU | 32 filter size,5x5 kernel |
| Conv2D –ReLU | 64 filter size,5x5 kernel |
| Conv2D –ReLU | 128 filter size,5x5 kernel |
| Flatten | 512 |
| Dense- Sigmoid | |
| Optimization / Loss | Adam / binary cross entropy |
| Epochs | 10 |
| Batch size | 32 |
| Learning rate | 0.001 |

An MPI cluster consisting of the considered nodes is created. The external IP addresses of these VM instances are used to establish a password-less ssh connection between them. An NFS (Network File System) directory was created and shared among all nodes to store the executable code and datasets. To facilitate distributed execution, each node in the cluster has an identical version of all software modules. The initial configuration of the cluster included 3 nodes: 1 master and 2 employees. Later, the system was expanded to accommodate 5, 7, 9, and 11 nodes.

## 5. Result Analysis

The results are presented in this section for the Data parallel model and model parallel model. The accuracy and timing analysis was done for the sequential and data parallel MPI-based framework in AWS Sage Maker Notebook instance for nodes varying from 2 to 32, as shown in Table 3. The maximum accuracy is obtained in the distributed framework.

$$FLOPS_{Conv2D} = 2\ x\ No\ of\ kernels\ x\ kernel\ size\ x\ \ Output\ Shape \quad (1)$$

$$FLOPS_{Fully\ connected\ layer} = 2\ x\ Input\ size\ x\ \ Output\ Shape \quad (2)$$

$$FLOPS_{Pooling\ \ layer} = Image\ height\ x\ image\ depth\ x\ \ image\ width \quad (3)$$

For Exa-scale Computing, the FLOPS are calculated depending on the layers based on equations 1 and 3. For the novel CNN model, the exascale computing FLOPS calculated are shown in Table 11. The total FLOP in a single node is 3.2E+15 FLOPS, and for the 5-node cluster etup, it is 1.28256E+16 FLOPS. Similarly, it can easily be scaled for a few hundred and thousand nodes in the AWS cloud, which has around 1.28E+19 FLOPS. For Exa-scale

Table 3. Performance measure

| Execution Model | Accuracy (%) | Time (s) |
|---|---|---|
| Input size | | 50x50 |
| Input size | 50x50 | |
| The proposed model (5 nodes) | 100 | 17.2157 |
| The proposed model (6 nodes) | 100 | 14.321 |
| The proposed model (7 nodes) | 100 | 11. 345 |
| The proposed model (8 nodes) | 100 | 9.456 |
| The proposed model (16 nodes) | 100 | 5.673 |
| The proposed model (32 nodes) | 100 | 2.456 |

Table 4. Exa-scale application size and its total FLOPS

| Layer | No. of Kernel | Kernel shape/Input | Output Shape | FLOPS |
|---|---|---|---|---|
| Input size | | | | 50x50 |
| Conv2D | 8000000 | 50x50 | 80000 | 3.2E+15 |
| Conv2D | 160000 | 5x5 | 160000 | 1.28E+12 |
| Conv2D | 320000 | 5x5 | 320000 | 5.12E+12 |
| Dense | 1 | 1280000 x1 | 1 | 2560000 |

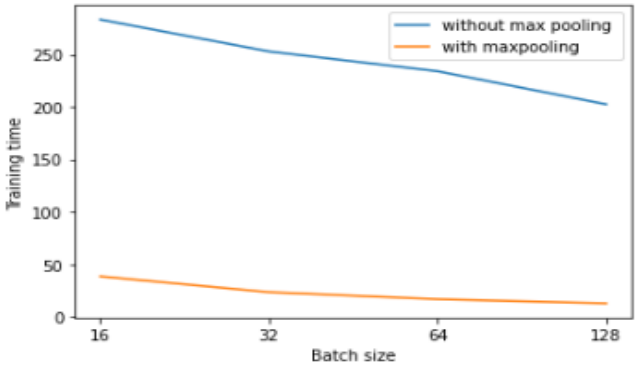


**Figure 3.** The average training time of all benchmark applications

Distributed Deep Neural Network Accelerator is a general framework that can be utilized for multiple domains to accelerate neural network processing in the distributed platform. The existing solutions need to provide an easily scalable framework. In the case of big data, a single processor is insufficient for data storage, and data needs to be distributed among nodes/processors. Distributed processing can lead to high complexity, and this framework helps reduce the model's complexity and processing time. AWS Sage maker helps to provide scalability, collection of rich inbuilt ML/DL libraries, easy hyperparameter tunning, accelerators, streamlining, etc. The main impact of CON-VLSTM-D-ACCEL is in the inherently distributed domains like medical, weather forecasting, banking, etc. The model makes the process simple using secure, customizable functions, parameters, and data since they can be stored in the different nodes and need not be shared with a central node for processing. The CNN model can be accelerated for exascale computing since the network operations are in the range of 1018 FLOPS based on the image size, filter, and kernel size.

## 5.1. ConvLSTM-D-Accel Data Parallel

Figure 4 illustrates the accuracy of individual benchmarks, while Figure 5 displays the training time in the distributed data-parallel mode. The distributed data-parallel model exhibits nearly double the speed of a CNN model operating on a single node. It is widely acknowledged that executing data parallelism on a single multi-core node result in both improved precision and faster processing. However, in the proposed ConvLSTM architecture, scalability necessitates distribution across multiple nodes.

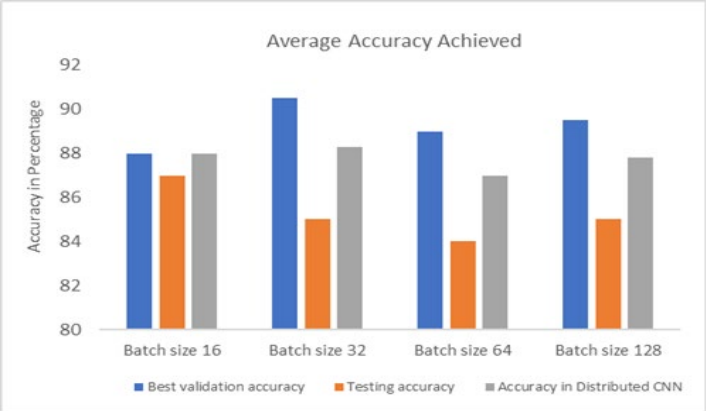


**Figure 4.** Data Parallel implementation accuracy vs batch size

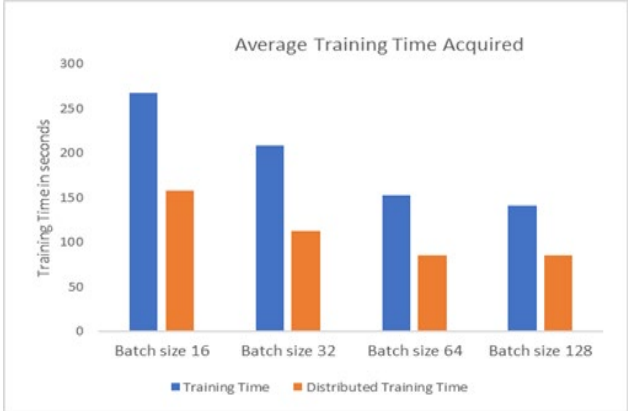


**Figure 5.** Data Parallel implementation training time vs batch size

Table 5. Comparison with Existing Frameworks

| Framework | Language | Parallelism | Open Source | Exa Scaling | Synchronization |
|---|---|---|---|---|---|
| Horovod | Python and C++ | Data Parallel | Yes | No | Model wise |
| Orca | Python | Data Parallel | Yes | No | Model wise |
| Tarantella | Python and C++ | Data Parallel | Yes | No | Model wise |
| Whale | Python | Data Parallel +Model Parallel | No | No | Model wise |
| Proposed Model | Python | Data Parallel | Yes | Yes | Epoch wise +layer Wise |

## 5.2. ConvLSTM-D-Accel Model Parallel

In Figure 6, details of the average accuracy achieved concerning ConvLSTM-D-Accel model parallel execution are shown. In Figure 8, a training time comparison is shown.

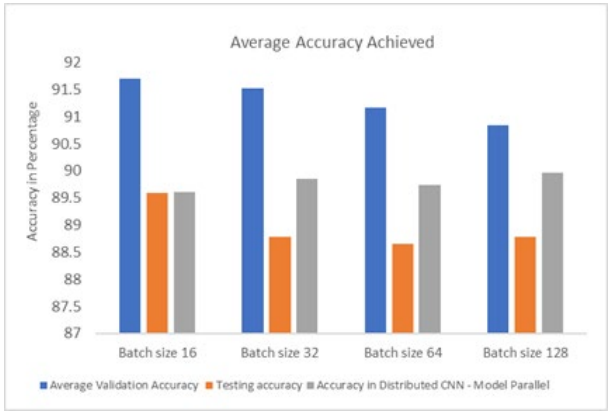


**Figure 6.** Model Parallel implementation accuracy vs batch size

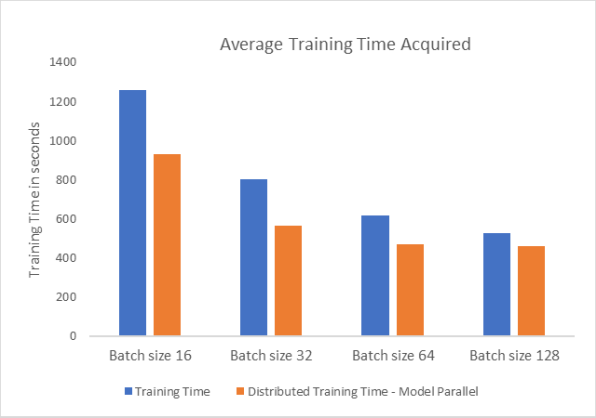


**Figure 7.** Model Parallel implementation training time vs size

## 5.3. ConvLSTM-D-Accel Scaling

In Figure 8, we see how the various configuration parameters affect the average accuracy over a range of batch sizes. The results show a small rise in precision after applying scaling.

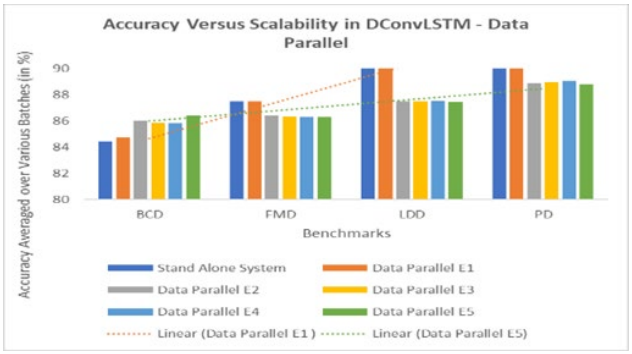


**Figure 8**. Data Parallel Model Accuracy Versus Scalability

## 6. Conclusion

CONVLSTM-D-ACCEL's usefulness resides in promoting this technique and making its application more accessible. Thus, research, education, banking, medical and business applications are possible benefits. This paper introduces CONVLSTM-D-ACCEL, an open-source and flexible Python framework for simple, accelerated neural network distribution. It allows users to spend less time implementing the distribution of more complicated models. In future versions, the model can be scaled for high-performance computing platforms GPU since it can be easily configured in AWS sage maker. Furthermore, the framework needs to be developed for modelling parallel deep learning algorithms in the future.

The ConvLSTM-D-Accel model employs a distributed convolutional neural network with LSTM for the automated adjustment of hyperparameters. On Google's cloud platform, virtual machines implement this approach. One master node and any number of worker nodes (two to eleven) are used in the experiment, with the latter's results subjected to careful analysis. The outcomes demonstrate a training time reduction of up to a factor of 2, with an accuracy rate of over 92% obtained. It's important to note that the distributed system in question works without stragglers or staleness in the context of this project.

## References

[1] Ravikumar, A, Sriraman, H, Sai Saketh, M, Lokesh, S, Karanam, A. Effect of neural network structure in accelerating performance and accuracy of a convolutional neural network with GPU/TPU for image analytics. PeerJ Computer Science. 2022; Vol. 8: pp. e909.

[2] Ravikumar, A, Sriraman, H, Sai Saketh, M, Lokesh. Identifying Pitfalls and Solutions in Parallelizing Long Short-Term Memory Network on Graphical Processing Unit by Comparing with Tensor Processing Unit Parallelism. Inventive Computation and Information Technologies; 2/3/2023; India. Springer; 2023. pp. 111–125.

[3] S. Harini and A. Ravikumar. Effect of Parallel Workload on Dynamic Voltage Frequency Scaling for Dark Silicon Ameliorating. International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2020; pp. 1012-1017

[4] Ravikumar, A, Sriraman, H. Real-time pneumonia prediction using pipelined spark and high-performance computing. PeerJ Computer Science. 2023; Vol. 9: pp. e1258.

[5] Ravikumar, A, Sriraman, H. Computationally Efficient Neural Rendering for Generator Adversarial Networks Using a Multi-GPU Cluster in a Cloud Environment. IEEE Access. 2023; vol. 11, pp. 45559-45571.

[6] Zagoruyko, S, Komodakis, N. Wide Residual Networks. Procedings of the British Machine Vision Conference 2016. pp. 87.1-87.12.

[7] Ravikumar, A. Non-relational multi-level caching for mitigation of staleness & stragglers in distributed deep learning. Proceedings of the 22nd International Middleware Conference, 1021. pp 15–16.

[8] Sriraman, H, Ravikumar, A, Keshwani, N. Malware Prediction Analysis Using AI Techniques with the Effective Preprocessing and Dimensionality Reduction. Innovative Data Communication Technologies and Application, 2022. pp. 153–169.

[9] Zhuang, D, Chang, J, Li, J. DynaMo: Dynamic Community Detection by Incrementally Maximizing Modularity, IEEE Transactions on Knowledge and Data Engineering, 2021.vol. 33, no. 5, pp. 1934–1945.

[10] Nasr, M, Shokri, R, Houmansadr, A. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning, IEEE Symposium on Security and Privacy (SP), IEEE Computer Society, 2016. pp. 739–753.

[11] Shokri, R, Stronati, M, Song, C, Shmatikov, V. Membership Inference Attacks Against Machine Learning Models, IEEE Symposium on Security and Privacy, 2017. pp. 3–18.