ASJET

Al-Salam Journal for Engineering and Technology
Journal Homepage:
http://journal.alsalam.edu.iq/index.php/ajest
e-ISSN: 2790-4822 p-ISSN: 2958-0862

Optimizing Training Time in Deep Learning Models Using Distributed Computing Techniques

Nasiba M. Abdulkarim[1], Lozan M. Abdulrahman[1], Taha A. Ababakar[2] and Omar M. Ahmed[3]*

[1]Information Technology Department, Duhok Polytechnic University, Duhok-KRG, 42001, Iraq.
[2]Computer Science Department, University of Zakho, Duhok-KRG, 42001, Iraq.
[3]Computer Information System Department, Duhok Polytechnic University, Duhok-KRG, 42001, Iraq.

*Corresponding Author: Omar M. Ahmed

DOI: https://doi.org/10.55145/ajest.2026.05.01.009
Received November 2025; Accepted January 2026; Available online Febuary 2026

ABSTRACT: Training deep learning models is often a time-consuming process, especially when hardware resources are tight. This bottleneck slows down experimentation and increases development costs. The primary goal of this study was to investigate whether distributed computing could cut down on training time while keeping model accuracy intact. To test this, we trained a Convolutional Neural Network (CNN) on the CIFAR-10 dataset using two different configurations: a standard single-device setup and a distributed synchronous setup powered by TensorFlow's MirroredStrategy. To ensure a fair comparison, both trials used the exact same architecture, hyperparameters, and preprocessing. The results were clear: distributed training reduced the total time by about 19.5%, with validation accuracy remaining nearly identical to the single-device approach. These findings suggest that distributed training is a practical way to accelerate deep learning workflows without sacrificing performance, serving as an efficient solution even in environments with limited hardware.

Keywords: Distributed Deep Learning, Convolutional Neural Networks, Data Parallelism, Model Scalability, Computational Efficiency

1. INTRODUCTION

Deep learning has become a core technology, in modern applications. The applications include image classification, speech recognition, medical diagnosis and natural language processing [1–3]. Over the ten years researchers have changed neural network designs and training methods. The changes have raised accuracy and overall performance [4]. However the improvements often bring computing load and longer training times. The higher load makes deep learning models cost more to build and harder to scale [5]. The model depth and dataset size grow. Training neural networks on the single computer becomes inefficient. Training neural networks on the single computer uses a lot of time. The large models need hours or days to converge. The large models cut the number of experiments a researcher can run. The large models slow the research progress. The problem appears in places, with computer resources.

The problem appears when the lab does not have the high-end GPUs or the specialized hardware [6]. As a result, reducing training time without sacrificing model performance has become an important research problem [7]. Distributed training has become a solution to the challenges. Distributed training divides the training workload across computing devices. Distributed training speeds up the model training a lot. Keeps the learning behavior the same [8]. Data-parallel training has gained adoption because Data-parallel training is simple and Data-parallel training works with the existing neural network architectures [9]. Popular deep learning frameworks now provide built-in tools that let distributed training run with changes, to the user code [10].

*Corresponding author: omar.alzakholi@dep.edu.krd

TensorFlow's MirroredStrategy uses a data-parallel method. The method lets several devices train the model copy while the parameters stay synchronized. The MirroredStrategy is common, in work. The MirroredStrategy gives learning and gives results that match single-device stochastic gradient descent [11]. In my experience synchronous distributed training can give speedups when the conditions are right especially when the communication overhead is low [12]. With those advantages many existing studies still focus on complex designs large hardware systems or GPU and TPU based setups [13]. Many existing studies give attention to evaluating distributed training in simpler and more accessible settings such as CPU only environments or small scale experimental setups. Understanding the performance benefits and the limitations of distributed training in distributed training conditions is important, for researchers and practitioners who work with resources [14]. We look at how distributed training works when we use TensorFlow's MirroredStrategy in a controlled experiment. We do not create a model design or a new training method.

This study tries to see the effect of distributed training on training time and on model accuracy. By keeping the data set the model design and the hyperparameters the same across all runs we make sure that any difference we see comes from training with TensorFlow's MirroredStrategy and not from other changes. We use the data set, the same model design and the same hyperparameters, for each training setup. That way we get an repeatable comparison [15]. The main objectives of this study are:

• to evaluate the reduction in training time achieved through synchronous distributed training using MirroredStrategy, and

• The goal is to check if model accuracy stays the same. The check compares model accuracy to standard single-device training, under the experimental conditions.

The contributions of this paper are listed below:

• A systematic comparison between single-device training and synchronous distributed training using TensorFlow MirroredStrategy.

• Quantitative analysis of training time and validation accuracy across different training configurations.

• Practical insights into the effectiveness of distributed training in low-resource, CPU-only environments.

The remainder of this paper is organized as follows. Section 2 reviews related work on distributed deep learning and training optimization. Section 3 describes the methodology of our work. Section 4 presents the experimental results and discussion. Finally, Section 5 concludes the paper.

2. RELATED WORKS

Several studies have investigated methods to reduce the training cost of deep learning models, particularly in distributed environments. Existing approaches can generally be grouped into model-level optimization, communication-efficient training, and system-level scheduling and resource management. This section summarizes these directions, highlights their trade-offs, and identifies the research gap addressed by this work.

2.1 Model-level optimization and sparsity-based methods

Model-level optimization techniques aim to reduce training cost by limiting unnecessary computation, often through sparsity, pruning, or selective updates. Prior studies have shown that introducing sparsity during training can significantly reduce computation while maintaining reasonable accuracy [16,17]. Other works focus on selectively updating model parameters or skipping less important operations to accelerate training [18]. While these approaches can be effective, they often require careful tuning and may introduce accuracy degradation when applied aggressively, especially for deeper models or complex datasets.

2.2 Communication-efficient and gradient-based approaches

Another major research direction focuses on reducing communication overhead in distributed training. Gradient compression, quantization, and reduced synchronization frequency are commonly used techniques to decrease data transfer between devices [19,20]. Several studies demonstrate that these methods can improve scalability and training speed, particularly in multi-node environments [21]. However, communication-efficient techniques may affect convergence stability and often depend on specific network conditions, making their performance less predictable across different setups.

2.3 Scheduling and system-level distributed training

System-level approaches improve resource use by optimizing task scheduling, workload balancing and hardware awareness during the training [22]. In my experience system-level approaches work well in the environments where devices have different speed or are sometimes not available. Previous work explored scheduling strategies to cut idle time and raise throughput [23,24]. System-level approaches are effective. System-level approaches often need special infrastructure or extra system-level complexity. That extra complexity can keep system-level approaches from being used in environments. To provide a clearer overview of existing approaches and highlight their key differences, Table 1 summarizes representative studies in the literature, comparing their main optimization strategies, hardware assumptions, and associated trade-offs, and positions the proposed approach within this context.

2.4 Summary and research gap

Overall, the studies discussed above demonstrate that significant training speedups can be achieved through model optimization, communication reduction, and scheduling strategies [16]–[24]. However, most of these approaches assume access to large-scale GPU clusters or specialized hardware, where communication and scheduling optimizations provide the greatest benefit.In contrast, less attention has been given to evaluating distributed training under low-resource settings, such as CPU-only environments, using practical and widely available tools. Addressing this gap is important for improving accessibility and reproducibility in deep learning research. Motivated by this observation, the present study evaluates TensorFlow's synchronous data-parallel training strategy (MirroredStrategy) as a practical baseline for accelerating training while preserving accuracy in constrained computational environments [25].

Table 1 Comparison of representative related work and the proposed approach

Ref.	Approach type	Main optimization	Typical hardware	Key trade-off
[16], [17]	Sparsity-based	Computation reduction	GPU / cluster	Possible accuracy loss
[18]	Selective updates	Compute efficiency	GPU	Tuning complexity
[19]–[21]	Gradient / communication	Communication cost	Multi-node systems	Convergence stability
[22]–[24]	Scheduling / system-level	Resource utilization	Heterogeneous clusters	System dependence
[25]	Synchronous data-parallel	Training time	CPU-only / low-resource	Limited scaling vs GPUs

3. METHODOLOGY

The methodology section that follows describes the Meteorological Data Lake and its ability to support such experiments, as well as laying out our experimental design. This approach is detailed later in this section; specifically, a distributed computing technique was added into CNN training. The methodology is developed to make all phases of the study, transparent and replicable, based on a rigorous approach, grounded in the experimental code implementation. The key aspects of the methodology consist of dataset selection and preprocessing, model architecture, distribution strategy, experimental setup, and performance evaluation. Figure 1 shows the schematic overview of the entire work-flow of this proposed distributed training framework.

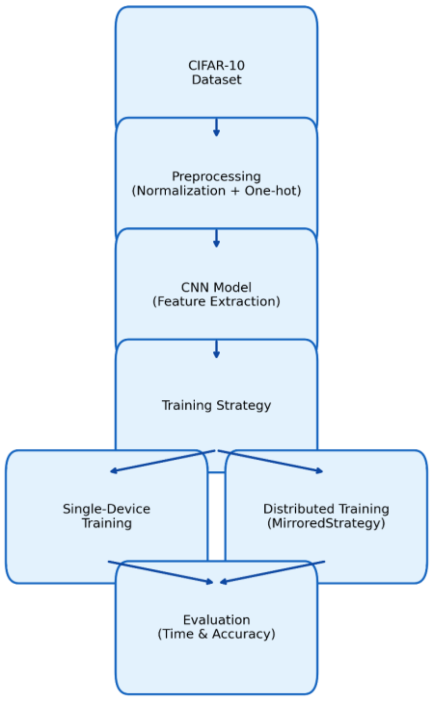
CIFAR-10 Dataset
Preprocessing (Normalization + One-hot)
CNN Model (Feature Extraction)
Training Strategy
Single-Device Training
Distributed Training (MirroredStrategy)
Evaluation (Time & Accuracy)

FIGURE 1 Overall workflow of the proposed distributed CNN training framework using single-device and distributed execution

3.1 DATASET AND PREPROCESSING

The CIFAR-10 image classification dataset was used for the experiments in this study, which is a widely-used benchmark. CIFAR-10 is composed of 60,000 color images of dimension 32×32 pixels evenly distributed into 10 object classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck). It is split into 50,000 training images and 10,000 test images to ensure a fair evaluation. This dataset was selected because it is a widely used benchmark for image classification and allows controlled and reproducible experiments. Its moderate size makes it suitable for evaluating training efficiency and runtime behavior, which is the main focus of this study, rather than maximizing classification accuracy.

Before training the model, the input data is preprocessed to normalize it and facilitate convergence. The pixel values of each image (originally in the range between 0 and 255) were normalized to the range between 0 to 1 in floating point with all values divided by 255. This is a common component of deep learning pipelines because it helps normalize gradient descent techniques and speed up training convergence. The data set was also split into training and validation sets. 20% of the training set was set aside for validation during training to check generalization and avoid overfitting.

Labels were also mapped from integer class identifiers to one-hot vectors with the TensorFlow package utility. This transformation is to keep in line with the categorical cross-entropy loss function applied during optimization, which makes model easier to learn class probabilities.

3.2 MODEL ARCHITECTURE

The CNN architecture used is deliberately a trade-off between computational efficiency and classification capacity, as our focus was on distributed training performance evaluation rather than optimal accuracy. A simple convolutional neural network was intentionally used to isolate the effect of distributed training on training time. Using a lightweight architecture reduces confounding factors related to model complexity and allows a fair comparison between single-device and distributed training setups. The architecture is divided into the following layers as shown in the code implementation:

Input Layer: It will accept batches of 32×32×3 images, where 3 is the number of channels.

Convolutional Layer 1: 32 3×3 filters with ReLU Rectification. Padding was adjusted to “same”, which would maintain the spatial dimensions after convolution. This is where low-level image features like edges and color gradients are recorded.

Local Pooling Layer Max-pooling layer 1: Squashes by performing 2×2 max pooling, resulting in half the spatial dimension with the most important features saved. This makes the model more computationally efficient and reduces overfitting.

Conv2: Further enriches feature representation with 64 filters of size 3×3 and the ReLU activation. Like the second convolutional layer, it has “same” padding which is used to keep input size the same.

Max-Pooling Layer 2: A second 2×2 max pooling layer to further decrease spatial resolution and computations.

Flatten Layer: Reshapes the tensor with multiple dimensions into a single dimension, which is needed for feeding it to fully connected layers.

Dense Layer 1: A dense (fully connected) layer with 64 neurons and ReLU activation, which allows for high-level, abstract feature learning.

The output layer is a dense layer with 10 neurons and softmax activation function that generates class probability distributions over the 10 possible CIFAR-10 categories.

This architecture was implemented in TensorFlow/Keras and compiled using the Adam optimizer with a learning rate of 0.001. The categorical cross-entropy loss function was selected due to its suitability for multi-class classification tasks, and model performance was tracked using the accuracy metric. The structure of the CNN used in this study is shown in Figure 2.

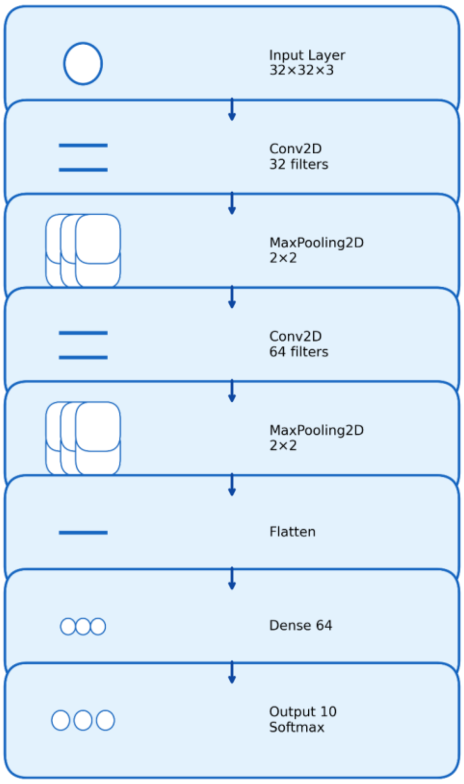

FIGURE 2 Architecture of the convolutional neural network used for CIFAR-10 image classification

3.3 DISTRIBUTED TRAINING STRATEGY

To investigate the impact of distributed computing on training efficiency, the study employed TensorFlow's tf.distribute.MirroredStrategy, a synchronous data-parallel strategy. This strategy is designed to replicate the model across multiple devices (e.g., CPUs or GPUs), splitting input batches and synchronizing updates during backpropagation. TensorFlow's MirroredStrategy was chosen because it provides synchronous data-parallel training with minimal implementation overhead and ensures training behavior equivalent to single-device stochastic gradient descent. This makes it suitable for controlled performance comparison.

The baseline training was performed with the default single-device execution, without distribution policy. In performing the experiment, training loop was surrounded by strategy scope function; all the variables of the model would be mirrored on all available devices, in this case. The gradients were calculated at every step of training, averaged over each replica, and applied synchronously to update all devices. This method drastically mitigates idle compute time, and exploits parallelism of hardware for training speedup.

The code implementation exactly mirrors this design, where the CNN model was constructed within the strategy scope for distribution. Batch size and epochs number were meanwhile synchronized in both setups to ensure fair comparison. In both cases, we set the batch size to 64 and trained for 10 epochs. The tf part is a synchronous data-parallel training Mirrored Strategy is illustrated in Figure 3.

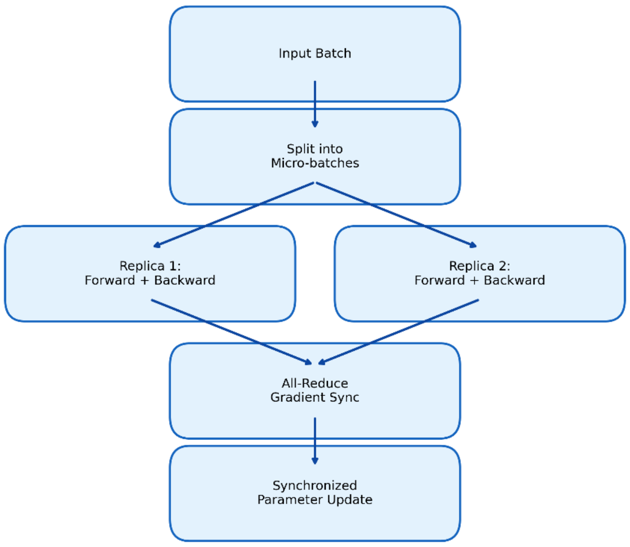
Input Batch
Split into Micro-batches
Replica 1: Forward + Backward
Replica 2: Forward + Backward
All-Reduce Gradient Sync
Synchronized Parameter Update

FIGURE 3 **Data-parallel distributed training workflow using synchronous gradient aggregation**

3.4 EXPERIMENTAL SETUP AND HYPERPARAMETERS

The Experiments were carried out on a machine with central processing unit (CPU). While TensorFlow's distribution strategies usually exhibit higher speedup when run on multiple GPUs, testing on CPUs enables the benchmarking to be controlled and showcases portability of this strategy across all hardware platforms. All experiments were conducted in a CPU-only environment to demonstrate the practicality of distributed training under limited hardware resources. This choice reflects realistic constraints faced by many researchers and institutions.

The system was implemented in TensorFlow to enable seamless integration with current deep learning pipelines. All random seeds were set in order to be able to reproduce the results. Several key features describe the experimental configuration:

- Architecture: TensorFlow using Keras for building, compiling, and training the model.
- Distribution Strategy: Compare baseline single-device running with the use of Mirrored Strategy.
- The batch size: of the mixed samples is set to 64.
- Epochs: 10 iterations over the entire set of training data.
- Hardware: CPU-based environment.
- Evaluation Metric: The validation accuracy on the 10,000 test images at the end of each epoch.

We also recorded the training duration of each experiment by logging the start and end time via Python 's on-board time module. This allowed to measure precisely execution time differences introduced through distributed training.

3.5 EVALUATION METRICS

We assessed the training schemes with the two metrics:

Train Time: measured in seconds; Train Time shows how long it takes to train for ten epochs. I look at Train Time to see the computer performance. Train Time also shows the speed up benefit of distributed training, on the workflow execution.

Accuracy on Validation: I track the accuracy on validation as a percent. I calculate the accuracy on validation after each epoch at the end of an epoch on the test set. I watch the performance on the validation set to make sure the improvement in training time is not, at the expense of model generalization.

Since the two metrics are adopted for purpose they illustrate by inherent they will be included through our objective work to evaluation work load on DC toward deep learning computation optimization. Efficiency is quantified by the training time, and the validation accuracy is ensured to prevent performance degradation.

3.6 REPRODUCIBILITY AND IMPLEMENTATION DETAILS

We implemented the method in Python using TensorFlow Keras API and we kept a codebase that is readable and reproducible. We put the functions that define the model the training and the evaluation into modules to make the code modular. We added pre-processing to the training pipeline using TensorFlow Datasets utilities. We built training scripts for both distributed and non-distributed runs. The unified training scripts allowed comparison, under the same experimental settings. We ran all experiments with TensorFlow. The same model architecture, dataset split, hyperparameters, and training procedure were used across all experiments to ensure a fair comparison. Training time and accuracy were measured consistently under identical conditions, enabling reproducibility of the reported results. By utilizing reproducible code-based methodology, the study guarantees transparency to practitioners who are planning to practice distributed computing in their workflows. The design can be easily extended to GPUs/TPUs, larger datasets and more complicated architectures which makes it a strong base for future study.

4. RESULTS AND DISCUSSION

The experimental results show that synchronous distributed training using MirroredStrategy leads to a noticeable reduction in training time compared to single-device training. Specifically, the distributed setup reduced the total training time by approximately 19.5%, while achieving nearly identical validation accuracy. The difference in final accuracy between the two setups remained below 1%, indicating that performance was preserved. One intention is to put the effect of distributed computing into consideration, i.e. how TensorFlow's Mirrored Strategy influenced both the training time and performance of your model while you developed a Convolutional Neural Network (CNN). Experiments were also configured to maintain the settings between the baseline and distributed setups relatively fixed in order to facilitate a fair comparison. In this talk, I will discuss how to understand these results in terms of the efficiency of deep learning and its implications.

4.1 TRAINING TIME ANALYSIS

The major result of the experiments is that the training time was cut when we use distributed computing. As shown in Table 2, the CNN model could be trained with no distribution for 10 epochs for 156.63 seconds and it was distributedly trained for only 126.12 seconds. This decrease translates into a training speed-up of roughly 19.5%. The observed speedup is mainly due to parallel data processing across multiple devices, which reduces the per-epoch computation time. Because synchronous updates are used, the learning behavior remains consistent with single-device stochastic gradient descent, explaining the similar accuracy trends observed across both training setups.

Table 2 Training time and validation accuracy with and without distribution

Method	Training Time (s)	Validation Accuracy
Without Distribution	156.63	0.6892
With Distribution	126.12	0.6907

The drop in training time reflects how performant it is to divide workloads onto different computing resources. Although we only had CPU hardware for the tested package in this work, we still have found a significant acceleration. This finding implies that parallelized training strategies can be beneficial even in small computational environments. One might also expect even greater speed-ups on multi-GPU/TPU setups that are popular in industry and research.

The gains are achievable when distributing the model on devices and updating between them at any step of training. Instead of serial processing a batch on one device, you distribute the workload and do it in parallel. This symmetry avoids idle time, and therefore utilization of all the computer muscles is improved, hence efficiency gain.

4.2 VALIDATION ACCURACY

The validation accuracy stayed almost constant after 10 epochs for both experiment configurations. Without distribution, the accuracy of 0.6892; with it, we can raise that to 0.6907. The slight discrepancy between the two values reassures that distributed training did not erode the model's generalization ability on held-out data.

This result is significant, because it verifies that (in principle) model performance need not be compromised to optimize models for computation through distribution. In practice, practitioners often worry that changing their training setup may introduce instabilities or affect convergence. However, these results show that synchronous data parallelism implemented with Mirrored Strategy matches accuracy but accelerates computation.

4.3 EPOCH-WISE PERFORMANCE TRENDS

An analysis of training and validation behavior with respect to epochs gives us further insight. In either setup, validation accuracy monotonously increased, reaching the ~0.69 limit around the 10 th epoch for the CNNs. The sequence of learning curves was almost identical in the two variants, hence supporting that decision making does not change the course of optimization. As example in Figure 4, also both training approaches show a well behaved behavior with respect to the validation accuracy over epochs.

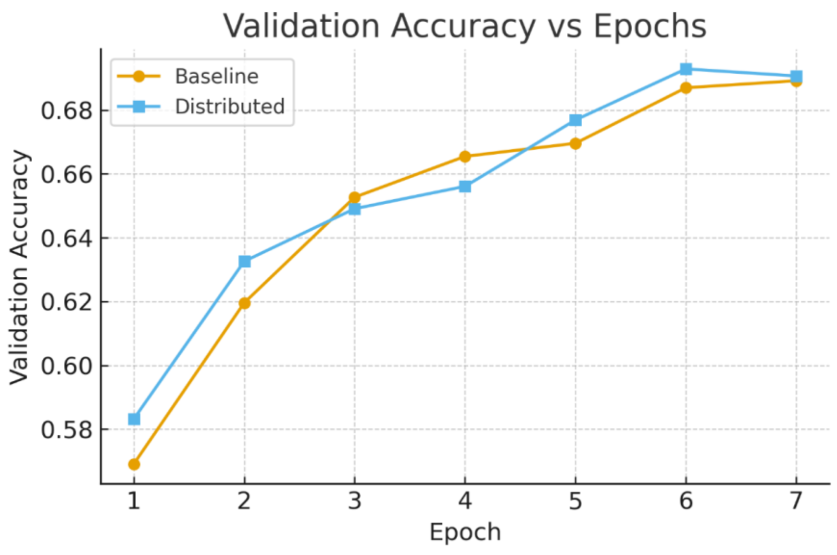

FIGURE 4 Validation accuracy curves for baseline and distributed training across seven epochs

That the learning dynamics of Adam optimizer can be stable under distributed training, when combined with categorical cross-entropy loss, is a phenomenon that holds across epochs. It further indicates that the averaging of gradients across devices is not incrementally washes out important updates such as weight quantization found in some asynchronous schemes. Figure 5 also shows that the validation loss curves provide evidence for both variants to converge smoothly during training.

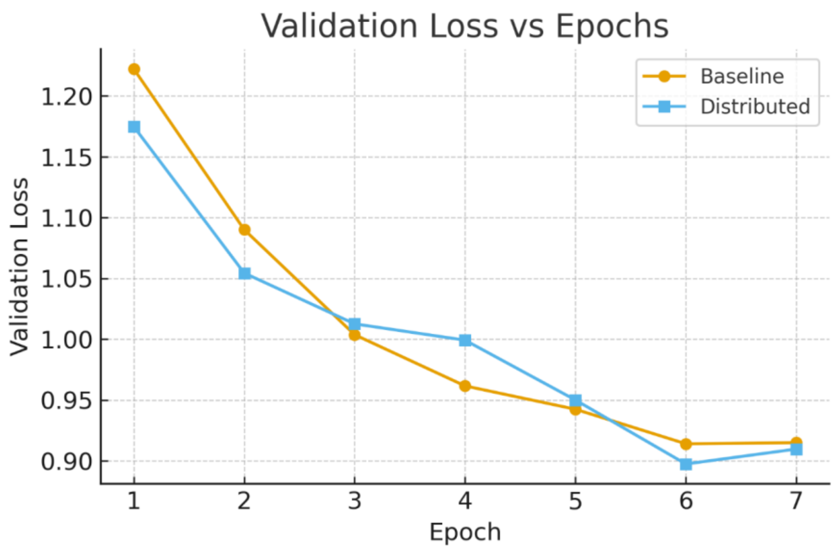

FIGURE 5 Validation loss curves for baseline and distributed training across seven epochs

4.4 IMPLICATIONS FOR DEEP LEARNING WORKFLOWS

The implications of our results for practitioners in deep learning are immediate. Training deep neural networks is time-consuming, especially on large datasets and architectures. These study results further demonstrate that distributed computing can accelerate training duration without sacrificing effectiveness and it will enable faster model development and iteration.

For academic researchers, shorter training means more hypotheses and configurations that can be tested within the same time budgets. For industrial purpose, it is important for enabling quick time to market and efficient resource usage of the distributed training as a tool to enhance productivity and minimize cost. These time savings, as seen here, are significant even in a CPU-only context, and can potentially snowball when applied to larger experiments.

4.5 LIMITATIONS OF THE CURRENT SETUP

While The benefits are clear in the results; however, one should bear in mind that in the sense of experimental conditions there are several limitations. The hardware configuration of this work is CPU only, so the parallelism cannot be very high. In modern deep learning, workflows utilize GPUs or TPUs that are specialized in matrix operations and can make better use for parallelization. That is, the Figure underestimates to very conservative degree what could be achieve when using distributed ML.

Also, the model implemented in this experiment was purposefully kept simple with two convolutional and two dense layers. For bigger and more complex models (such as transformerbased networks, ResNet, DenseNet), which are not possible to train on the baseline due to prolonged running times, the advantage of parallelization likely increases along with the complexity of these models.

4.6 BROADER CONTEXT OF DISTRIBUTED TRAINING

These findings are consistent with the existing work on distributed deep learning that often cites faster training (training time) as a main benefit of parallelized training schemes. Previous works show that such frameworks as Horovod, PyTorch Distributed Data Parallel (DDP), and TensorFlow's built-in strategies can scale to hundreds of devices with near-linear speed-up. The present work adds to it by (1) showing that these principles also hold in a reproducible small-scale comparison on readily available hardware with a widely used benchmark dataset, and (2) evaluating their effects on every level of the model.

We note that distribution strategy did not adversely affect accuracy, which essentially confirms synchronous data parallelism is a robust way to match single device training performance. On the other hand, asynchronous algorithms might expect to be faster but are prone to cause noise in convergence and need fine tuning. Therefore, the current findings lend support to the notion that simultaneous administration is a practical and efficient option for clinicians.

4.7 FUTURE DIRECTIONS

There are several issues the findings raise for future research. This is the hardware variant of these experiments and we are done experiment on GPU so this would represent the actual improvement that can be fair up to here. On the other hand, it may also inspire the understanding of its scalability on harder settings such as deeper/larger datasets (e.g., ImageNet). Moreover, exploring hybrid methods, i.e., combining data and model parallelism can achieve significant performance boost for services with billions of parameters.

5. CONCLUSIONS

This study investigated at how training changes the time it takes to train a deep learning model while keeping the classification accuracy the same. We used TensorFlow's MirroredStrategy to compare single-device training, with distributed training. We saw that distributed training reduced the training time a lot and gave identical accuracy. The results show that distributed training can be used in an practical way even in CPU-only environments. In my view this method helps researchers who have limited resources. Researchers can obtain efficiency gains without changing the model architecture or using complex optimization techniques. The study uses a CNN model, the CIFAR-10 dataset and a CPU only setup. The simple CNN model, the CIFAR-10 dataset and the CPU setup may limit scalability compared to GPU or TPU systems. Future work will extend the evaluation to models, larger datasets and more advanced hardware configurations. Future work will also explore distributed training strategies. Overall, the findings confirm that distributed training is an effective and accessible approach for reducing training time while preserving model performance.

FUNDING

None

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their efforts.

CONFLICTS OF INTEREST

The authors declare no conflict of interest

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.
[2] Y. Bengio, I. Goodfellow, and A. Courville, Deep learning, vol. 1. MIT press Cambridge, MA, USA, 2017.

[3] T. Brown et al., “Language models are few-shot learners,” Adv Neural Inf Process Syst, vol. 33, pp. 1877–1901, 2020.

[4] D. Patterson et al., “The carbon footprint of machine learning training will plateau, then shrink,” Computer (Long Beach Calif), vol. 55, no. 7, pp. 18–28, 2022.

[5] M. Li, D. G. Andersen, A. Smola, and K. Yu, “Communication efficient distributed machine learning with the parameter server,” Adv Neural Inf Process Syst, vol. 27, 2014.

[6] H. M. L. C. C. Power and D. A. I. Progress, “AI and Compute,” 2022.

[7] M. Abadi et al., “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” arXiv preprint arXiv:1603.04467, 2016.

[8] P. Goyal et al., “Accurate, large minibatch sgd: Training imagenet in 1 hour,” arXiv preprint arXiv:1706.02677, 2017.

[9] “tf.distribute.Strategy API documentation,” TensorFlow. Accessed: Nov. 13, 2025. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/distribute/Strategy

[10] A. Sergeev and M. Del Balso, “Horovod: fast and easy distributed deep learning in TensorFlow,” arXiv preprint arXiv:1802.05799, 2018.

[11] M. Li, T. Zhang, Y. Chen, and A. J. Smola, “Efficient mini-batch training for stochastic optimization,” in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 661–670.

[12] D. Filters’Importance, “Pruning filters for efficient convnets,” arXiv preprint arXiv:1608.08710, 2016.

[13] A. Paszke et al., “Pytorch: An imperative style, high-performance deep learning library,” Adv Neural Inf Process Syst, vol. 32, 2019.

[14] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in Proceedings of COMPSTAT’2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers, Springer, 2010, pp. 177–186.

[15] Y. Bengio, “Practical recommendations for gradient-based training of deep architectures,” in Neural networks: Tricks of the trade: Second edition, Springer, 2012, pp. 437–478.

[16] L. Mai, G. Li, M. Wagenländer, K. Fertakis, A.-O. Brabete, and P. Pietzuch, “{KungFu}: Making training in distributed machine learning adaptive,” in 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 2020, pp. 937–954.

[17] J. G. Pauloski, Z. Zhang, L. Huang, W. Xu, and I. T. Foster, “Convolutional neural network training with distributed K-FAC,” in SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2020, pp. 1–12.

[18] H. You et al., “Drawing early-bird tickets: Towards more efficient training of deep networks,” arXiv preprint arXiv:1909.11957, 2019.

[19] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen, “Rigging the lottery: Making all tickets winners,” in International conference on machine learning, PMLR, 2020, pp. 2943–2952.

[20] S. Horvóth, C.-Y. Ho, L. Horvath, A. N. Sahu, M. Canini, and P. Richtárik, “Natural compression for distributed deep learning,” in Mathematical and Scientific Machine Learning, PMLR, 2022, pp. 129–141.

[21] Y. Ko, K. Choi, H. Jei, D. Lee, and S.-W. Kim, “ALADDIN: Asymmetric centralized training for distributed deep learning,” in Proceedings of the 30th ACM International Conference on Information & Knowledge Management, 2021, pp. 863–872.

[22] L. Fournier and E. Oyallon, “Cyclic Data Parallelism for Efficient Parallelism of Deep Neural Networks,” arXiv preprint arXiv:2403.08837, 2024.

[23] G. Cruciata, L. Cruciata, L. Lo Presti, J. van Gemert, and M. La Cascia, “Learn & drop: fast learning of cnns based on layer dropping,” Neural Comput Appl, vol. 36, no. 18, pp. 10839–10851, 2024.

[24] M. A. Raihan and T. Aamodt, “Sparse weight activation training,” Adv Neural Inf Process Syst, vol. 33, pp. 15625–15638, 2020.

[25] Y. Ding et al., “Distributed optimization over block-cyclic data,” in Proceedings of the 6th ACM International Conference on Multimedia in Asia Workshops, 2024, pp. 1–6.