Group 20: A Comparison of Different Deep Learning Frameworks in a Standalone and Distributed Sense.

Introduction

The rapid advancement in artificial intelligence and machine learning, especially deep learning, has spurred extensive research and development in optimizing computational frameworks. Among these, TensorFlow, PyTorch, and Horovod have emerged as leading platforms, each with unique strengths and applications in various deep learning tasks. This report investigates the performance of these frameworks in both standalone and distributed settings, focusing particularly on image classification tasks using well-known models like VGG16 and ResNet50 trained on the CIFAR10 dataset. This comparison is vital as it provides insights into the practical applications of these frameworks in real-world scenarios where computational efficiency, scalability, and resource utilization are critical. Understanding these aspects helps in selecting the appropriate framework based on specific project needs, such as training time, hardware availability, and the complexity of the task at hand.

Literature Survey

"Large scale performance analysis of distributed deep learning frameworks for convolutional neural networks" - This article discusses the performance of distributed deep learning frameworks like Horovod, PyTorch-DDP, and DeepSpeed, particularly focusing on their synchronous communication and data-parallel functionality. It provides detailed benchmarks and insights into the scalability and efficiency of these frameworks when applied to training convolutional neural networks on large-scale datasets.[1]

"Reveal training performance mystery between TensorFlow and PyTorch in deep learning" - This paper delves into the specific performance characteristics of TensorFlow and PyTorch, comparing their efficiencies and identifying the scenarios where one may outperform the other. It's valuable for understanding the nuanced differences in how these frameworks handle deep neural network training.[2]

"Horovod: fast and easy distributed deep learning in TensorFlow" - Published on Arxiv, this paper introduces Horovod, a tool designed to enhance the ease and speed of distributed deep learning training in TensorFlow. It discusses the architecture of Horovod, its integration with

TensorFlow, and its impact on improving the scalability of training processes across multiple GPUs.[3]

"Distributed Deep Learning with Horovod" - Available on the NVIDIA developer site, this resource outlines practical implementations of Horovod with TensorFlow, PyTorch, and MXNet, explaining the strategies used to maximize training performance when scaling up to hundreds of GPUs. It provides practical insights into the deployment of Horovod in various industry applications.[4]

"DLBench: a comprehensive experimental evaluation of deep learning frameworks" - This study benchmarks several deep learning frameworks, including TensorFlow and PyTorch, across different hardware configurations and datasets. It offers a broad view of how these frameworks perform under various conditions, providing a well-rounded perspective on their capabilities and limitations.[5]

Methodology

We make comparisons between the Tensorflow, Pytorch, and Horovod frameworks for image classification tasks. We compare performance and results in a standalone and distributed setting. We use 2 famous pretrained models for the image classification task – VGG16 and ResNet50. Both these models have been trained on the Imagenet dataset. We use transfer learning to train these models on the CIFAR10 dataset.

First, we compare these frameworks without distributed training, we train each of our 2 models on the dataset using the 3 frameworks – Tensorflow, Pytorch, and Horovod and note down results such as time taken per epoch, GPU Utilization, and accuracy.

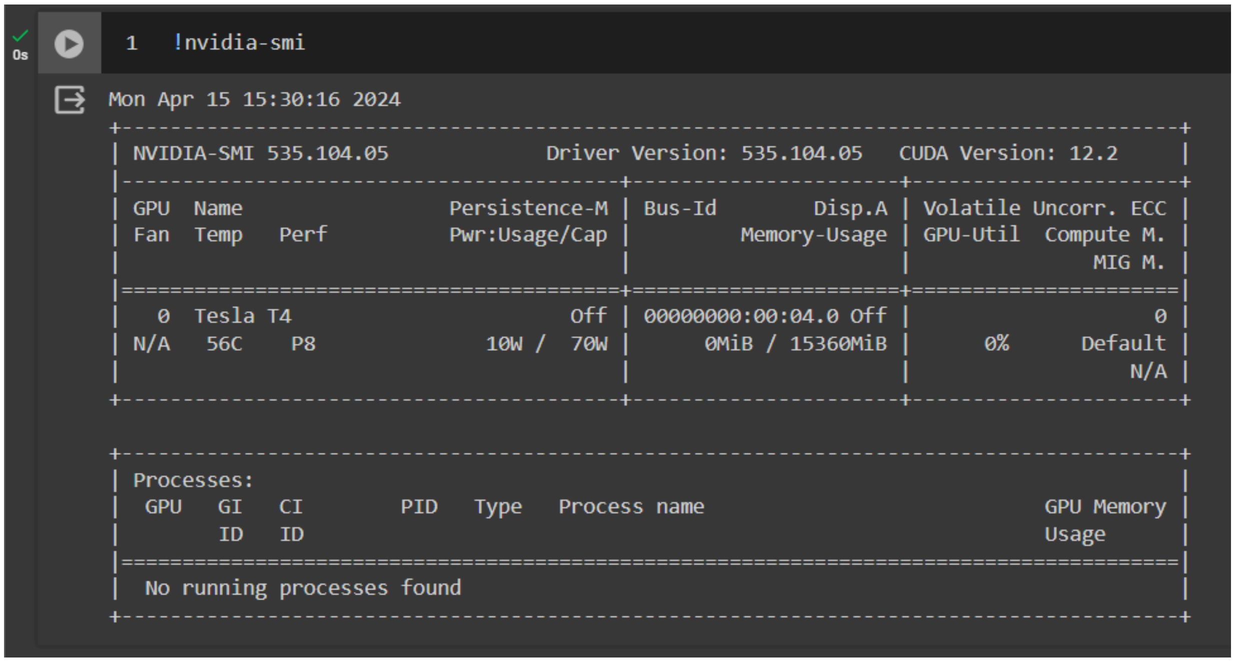
After this, we make the comparison between Pytorch and Tensorflow using Data Parallelization. We make use of the Mirrored Strategy in Tensorflow to accomplish this. The Mirrored Strategy is a form of synchronous distributed training where the model is replicated across all GPUs and each GPU is given a section of the batches on which to compute forward and backward gradients. After all GPUs finish computing the gradients, the model weights are updated simultaneously. This ensures that each GPU has the same model for training. We make use of the Distributed Data Parallel Strategy to implement Data Parallelism in Pytorch. It has a similar working to the Mirrored Strategy in Tensorflow. We perform these data parallelization tasks using both our models.

Next, we moved on to model parallelism. Using this technique, we split our model into two parts and distributed each part to a GPU. We did this for the ResNet50 as well as the VGG16 models and computed the results.

Experimental Setup

We made use of the Google Colab environment as well as the ASU Research Supercomputer (Sol). We used the Google Colab environment to run our standalone tests while we used Sol to perform our distributed training analysis using multiple GPUs.

We made use of the Tesla T4 GPU provided by Google Colab for our standalone tests. We can see in the image below that we use CUDA Version 12.2 and we have a GPU size of 15360 MiB which is equivalent to 15GiB.

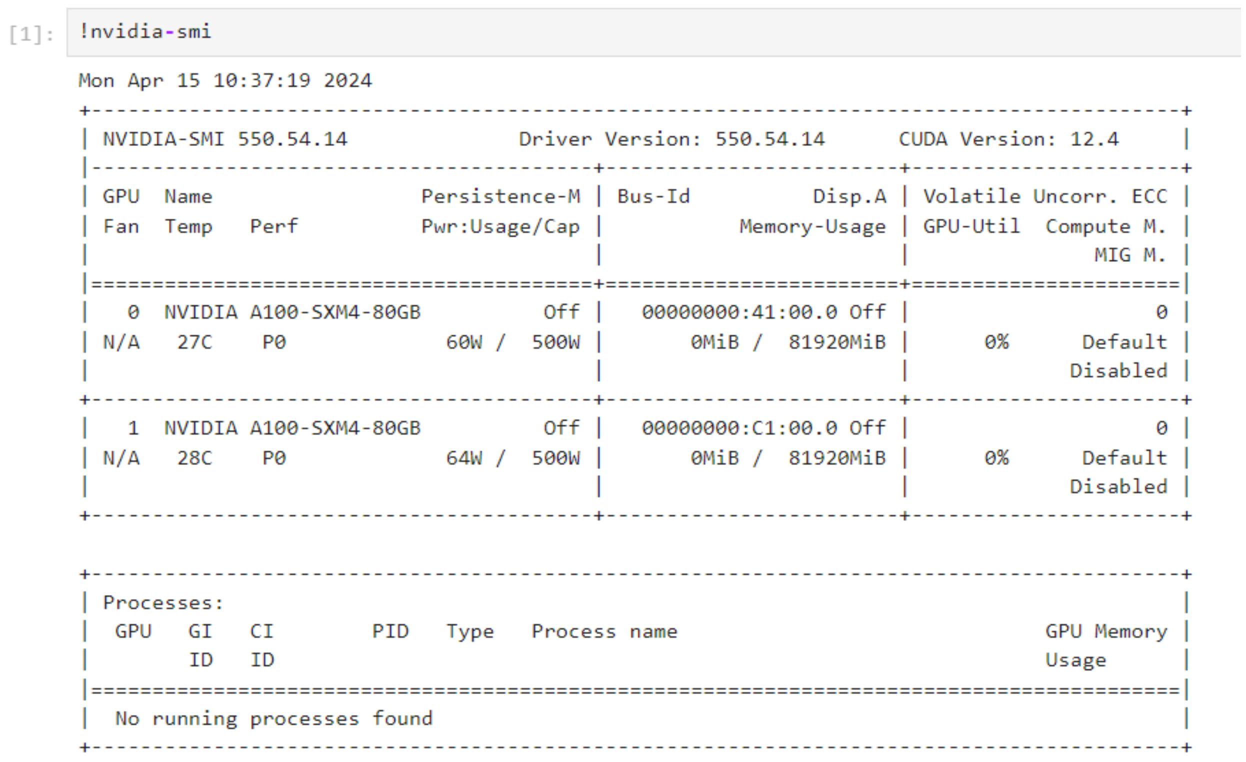


```
!nvidia-smi
```

```
Mon Apr 15 15:30:16 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   56C    P8              10W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

We made use of 2 A100 GPUs on Sol for the distributed training tests. As seen in the picture below, we are allocated 81920MiB in each chip which is equivalent to 8GiB. We are using CUDA version 12.4 in Sol.



```
[1]: !nvidia-smi
```

```
Mon Apr 15 10:37:19 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.14              Driver Version: 550.54.14      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA A100-SXM4-80GB          Off |   00000000:41:00.0 Off |                    0 |
| N/A   27C    P0             60W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA A100-SXM4-80GB          Off |   00000000:C1:00.0 Off |                    0 |
| N/A   28C    P0             64W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

To make a fair comparison we try to keep parameters consistent across all models, frameworks, and runs as shown in the table below.

Parameter	**Value**
Number of Epochs	10
Learning Rate	0.001
Batch Size	64
Optimizer	Adam
Loss Function	Categorical Cross Entropy

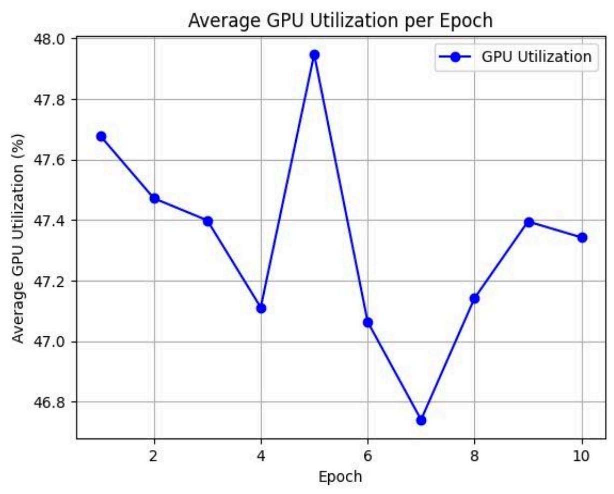
Data

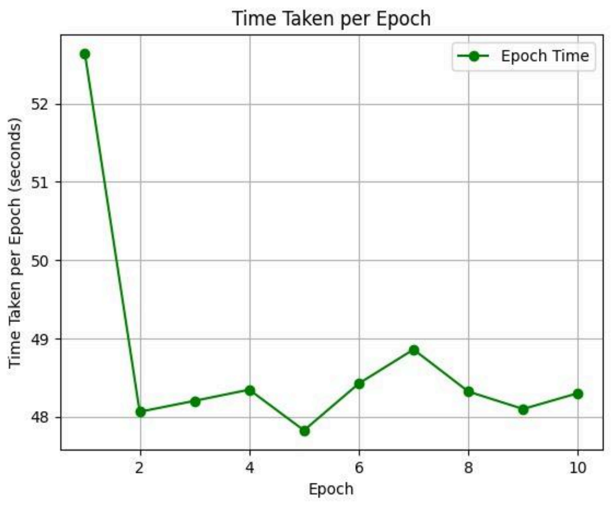
We made use of the CIFAR10 dataset to train our models and compare results. The CIFAR10 dataset is a very famous dataset for Computer Vision tasks. It consists of 60,000 colored images with 10 image classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.). It consists of 50,000 training images and 10,000 testing images. Each image has a dimension of 32 x 32 x 3.

Results

We are comparing the frameworks based on the time taken to run each epoch as well as the average GPU utilization per epoch. The images below show the graphs and values in detail. The table provided at the end summarizes these results and helps make a better comparison between frameworks without needing to flip across different implementations.

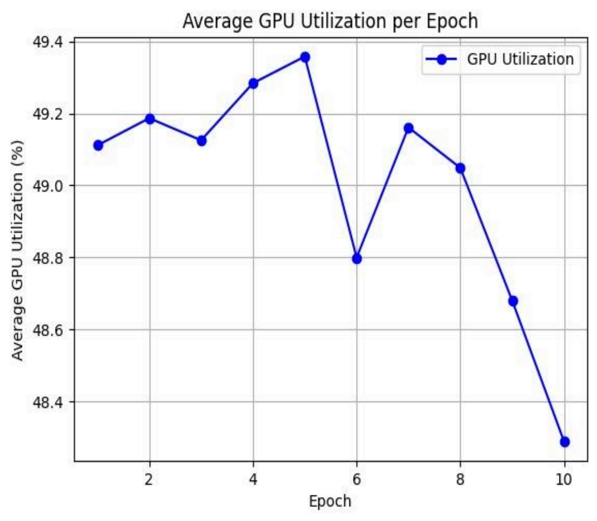
Results for Tensorflow framework with VGG16

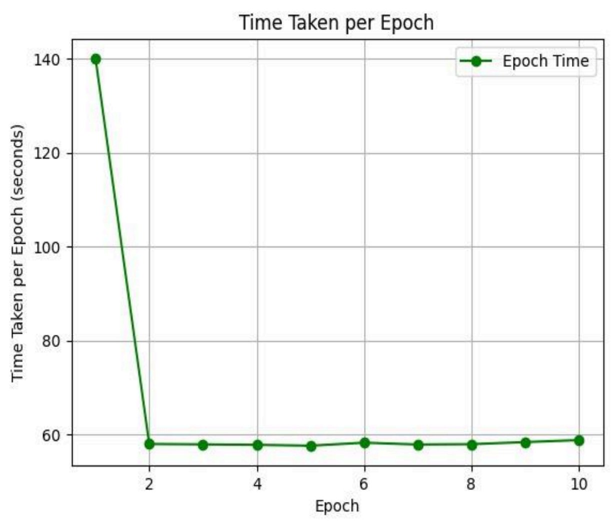




We can see that the average time per epoch is 48.6 seconds and the average GPU utilization is 47.4%.

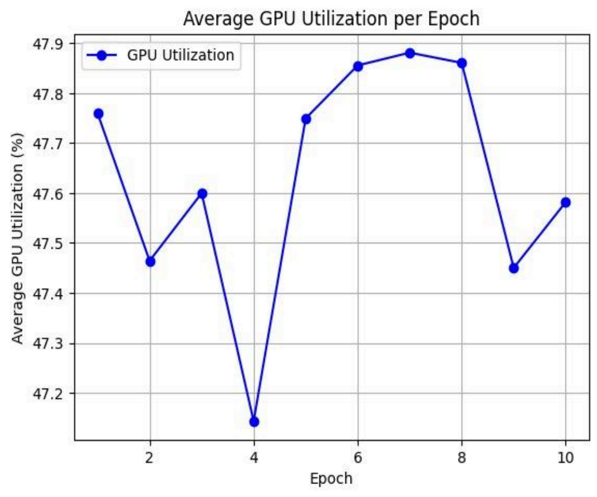
Results for Tensorflow framework with ResNet50

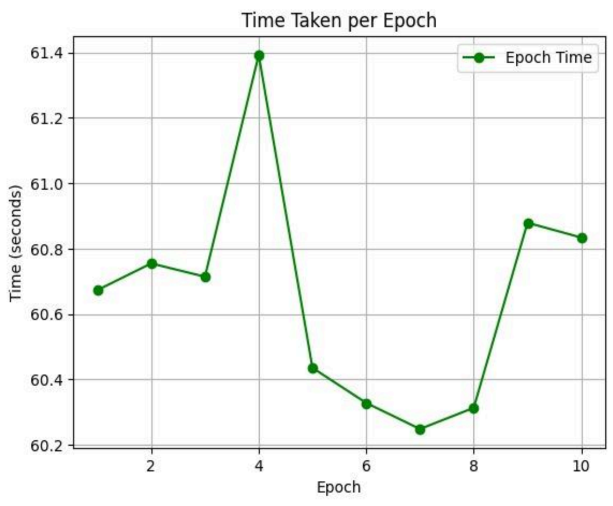




We can see that the average time per epoch is 57.6 seconds and the average GPU utilization is 48.9%.

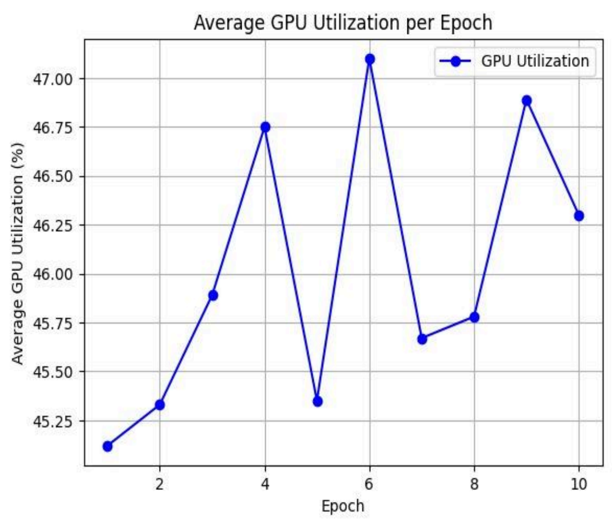
Results for PyTorch framework with ResNet50

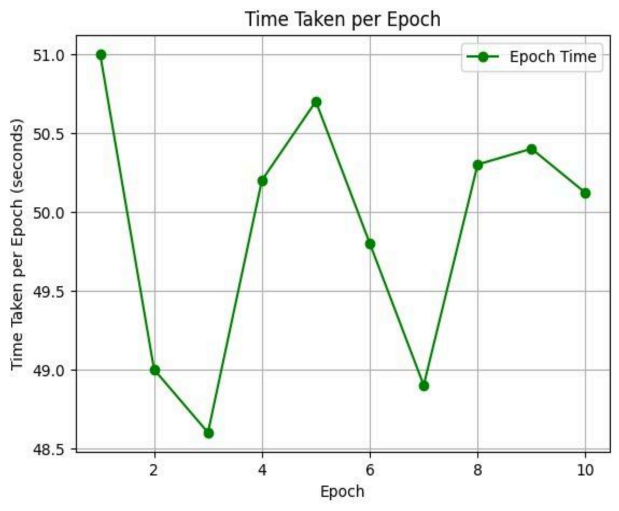




We can see that the average time per epoch is 60.6 seconds and the average GPU utilization is 47.5%.

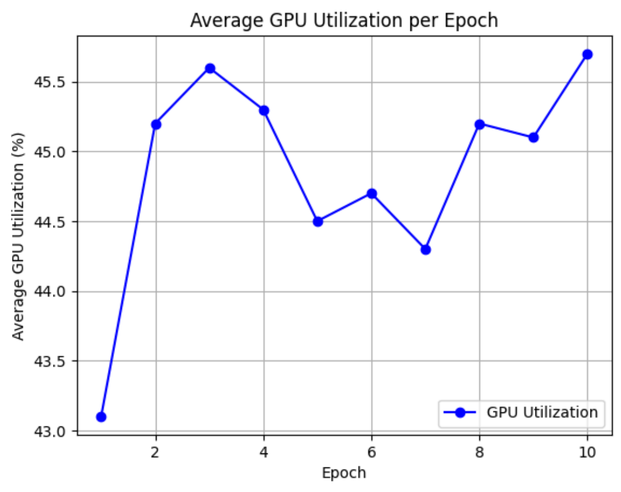
Results for PyTorch framework with VGG16

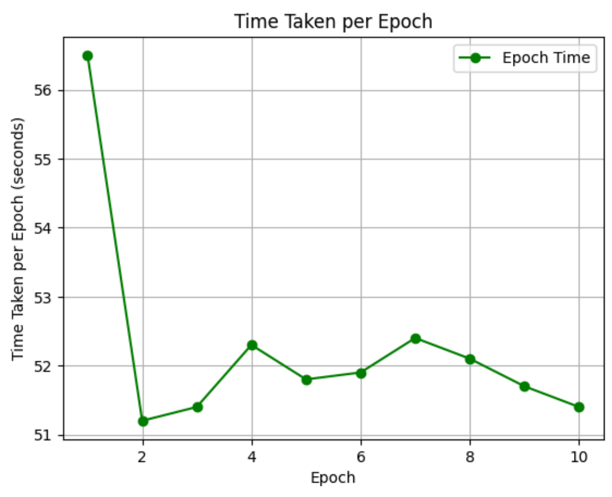




We can see that the average time per epoch is 49.8 seconds and the average GPU utilization is 46.1%.

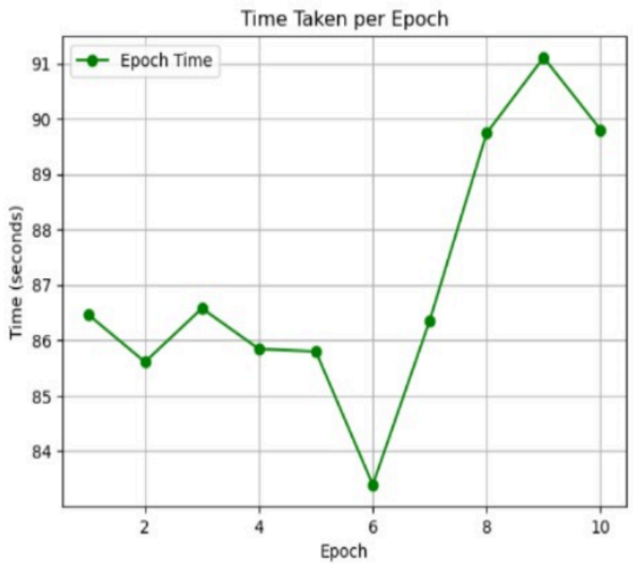
Results for Horovod Framework with VGG16





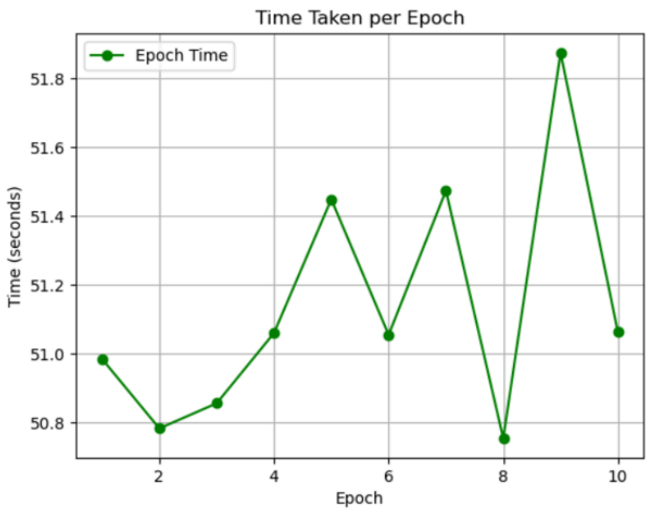
We can see that the average time per epoch is 52.1 seconds and the average GPU utilization is 44.5%.

Results for Model Parallelism with PyTorch



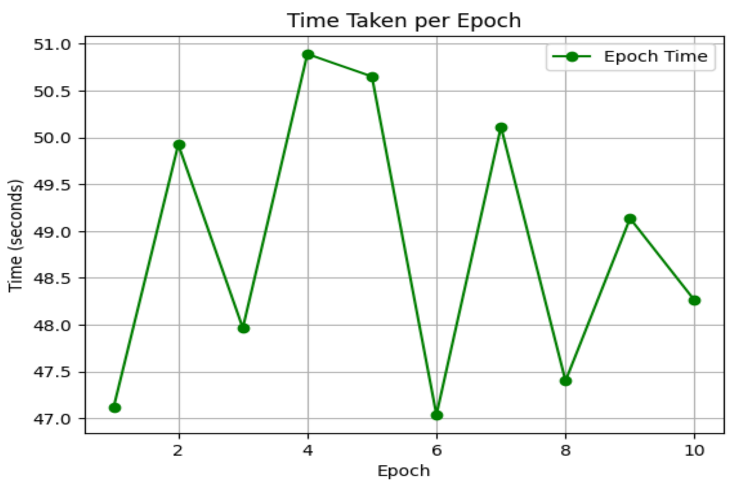
We can see that the average time per epoch is 87.1 seconds.

Results for PyTorch Data Parallelism



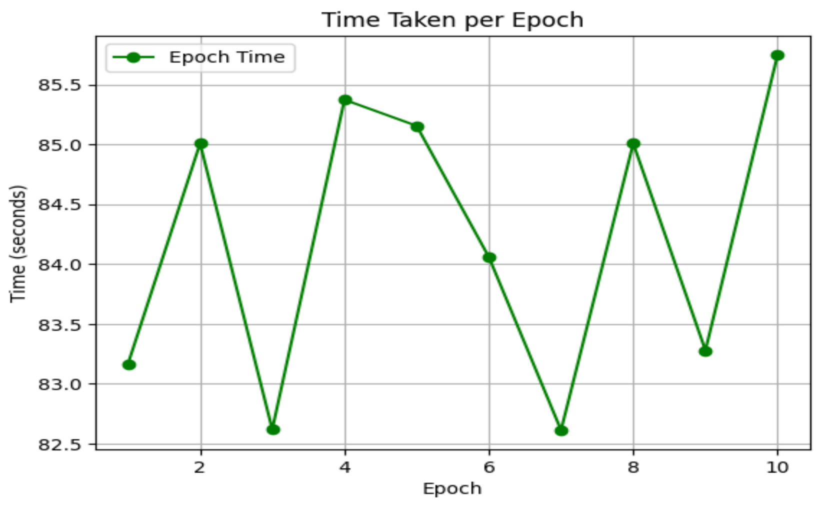
We can see that the average time per epoch is 51.1 seconds.

Data Parallelism in Tensorflow



We can see that the average time per epoch is 48.9 seconds.

Model Parallelism with TensorFlow



We can see that the average time per epoch is 84.2 seconds.

Comparison Table

Architecture	Average Time Taken Per Epoch	Average GPU Utilization
Standalone Tensorflow framework with VGG16	48.6 seconds	47.4%
Standalone Tensorflow framework with ResNet50	57.6 seconds	48.9%
Standalone PyTorch framework with VGG16	49.8 seconds	46.1%
Standalone PyTorch framework with ResNet50	60.6 seconds	47.5%
Standalone Horovod Framework with VGG16	52.1 seconds	44.5%
Model Parallelism with PyTorch and ResNet50	87.1 seconds	7.3%
Model Parallelism with Tensorflow and ResNet50	84.2 seconds.	9.6%
Data Parallelism in Tensorflow and ResNet50	48.9 seconds	18.2%
Data Parallelism in Pytorch and ResNet50	51.1 seconds	14.3%

Some insights we gained were:

1. Both Tensorflow and Pytorch were very competitive in all the tests we ran but TensorFlow seems to be slightly more efficient as compared to Pytorch.
2. We found both Time Taken per epoch as well as GPU Utilization per epoch to be more efficient for the VGG16 model compared to the ResNet50 models.

3. Model Parallelism performed worse as compared to the standalone models. This could be due to the small size of the models and the entire model being able to fit on a single GPU chip. The benefits of model parallelism did not outweigh the overhead costs.
4. The help of data parallelism helped reduce the time per epoch significantly but we were unable to achieve high GPU utilization values.

Conclusion

In conclusion, our comparative analysis between TensorFlow and PyTorch in both standalone and distributed settings using the ResNet50 and VGG16 models trained on the CIFAR-10 dataset has provided valuable insights into the strengths and trade-offs of each framework.

In the standalone sense, we found that both Pytorch as well as Tensorflow are very competitive and gave almost identical results, Tensorflow was marginally better than Pytorch with slightly lower training times per epoch as well as slightly higher GPU utilizations. We found that using Horovod without implementing distributed training proves to be counterproductive and increases the training time while reducing GPU utilization.

While talking about data parallelization, we found that the time taken per epoch decreased a significant amount for both the Pytorch as well as Tensorflow frameworks.

While comparing results using model parallelization, we found that our time taken per epoch and GPU utilization were both worse as compared to the standalone training technique. This implies that our model was small enough to fit in our GPU and using model parallelization just increased the overhead of communication while offering no real benefit in our case.

Ultimately, the choice between TensorFlow and PyTorch depends on factors such as project requirements, development preferences, and available resources. Both frameworks continue to evolve and innovate, driving advancements in deep learning research and applications. By leveraging the strengths of each framework, researchers and practitioners can effectively tackle a wide range of deep learning tasks and contribute to the advancement of artificial intelligence.

Future Work

There is scope for a lot more comparisons to get a better idea of the capabilities of each framework. Some of the ideas we would like to try are as follows:

1. Using asynchronous distributed training instead of synchronous distributed training.

2. Using multiple nodes on the machines instead of a single node.

3. Comparison using Pipeline Parallelism.

4. Using a model whose size is greater than the size of our GPU.

5. Using more datasets to compare results and find insights based on factors such as dataset size and attribute types and ranges.

6. Comparing the frameworks based on other Deep Learning Tasks such as Natural Language Processing, Generative Adversarial Network, Recommendation Systems, etc.

7. Comparing the effect of parameters such as batch size and Optimizer used on our models.

References

[1] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," arXiv preprint arXiv:1802.05799, 2018. [Online].

[2] S. Pumma et al., "Large scale performance analysis of distributed deep learning frameworks for convolutional neural networks," Journal of Big Data, vol. 7, no. 1, p. 88, 2020. [Online].

[3] S. Pumma et al., "Performance of distributed TensorFlow and Horovod in ML training tasks," presented at the GTC 2020, [Online].

[4] "Reveal training performance mystery between TensorFlow and PyTorch in deep learning," in Proceedings of the IEEE International Conference on Big Data Analysis, 2019. [Online].

[5] "DLBench: a comprehensive experimental evaluation of deep learning frameworks," in Proceedings of the IEEE Symposium on Benchmarking, Measuring, and Optimization, 2020.