Assignment II - Distributed Data Parallel Training

Group V
Badri Narayanan Murali Krishnan
Shreyas Subramanian
Rohan Purandhar
FNU Nevin John Selby
bmuralikrish@wisc.edu, ss7@wisc.edu
purandhar@wisc.edu, nselby@wisc.edu

Github Link: https://github.com/MBadriNarayanan/CS744

February 23, 2024

1 Part One: Training VGG-11 on CIFAR-10

1.1 Solution

We finalized the foundational training scripts. We accomplished the initial training process involving forward and backward passes, loss calculation, and optimizer setup in the *main.py* file. Additionally, we logged the loss of every 20 batch iteration. After running 40 mini-batch iterations and disregarding the initial one, we computed the average runtime which was found to be 1.477 seconds. The training loss was 4.652, the test loss was 3.529, and the test accuracy was 13.1%.

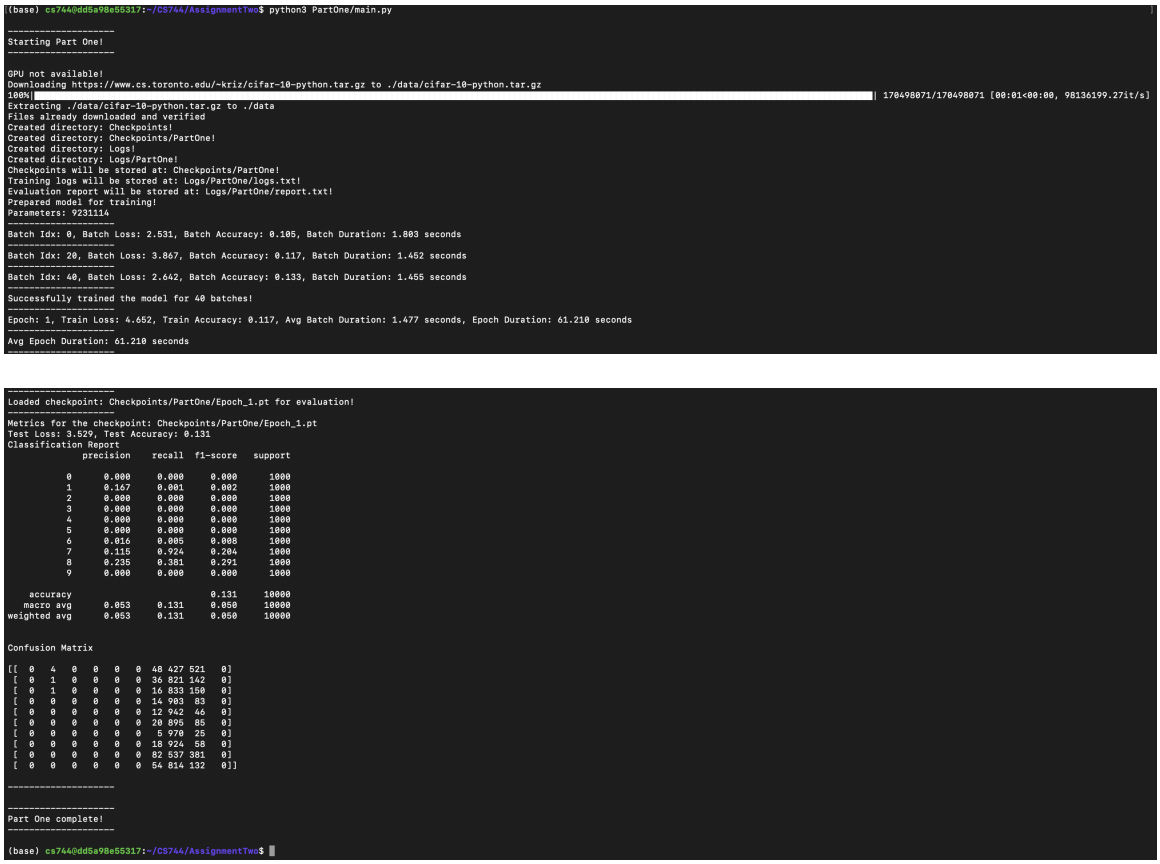

Figure 1: Output snapshots for Part One.

2 Part Two: Distributed Data Parallel Training

2.1 Part 2a: Sync gradient with gather and scatter call using Gloo backend

2.1.1 Solution

The training process was completed successfully, producing the expected results. The final average train loss value was 5.784, the average test loss 2.753, while the test accuracy was 11.9% - slightly different from part 1 but still within the anticipated range. Table 1 denotes the avg time it took to run 40 mini batch iterations and the epoch duration across the different nodes with a batch size of 64. From the table, we see that the average batch duration was 1.55 seconds with an average epoch duration 47.894.

Table 1: Metrics across the different nodes for Part Two Task One.

Node	**Avg Batch Duration**	**Epoch Duration**
Master Node	1.155	47.894
Node 1	1.155	47.893
Node 2	1.155	47.892
Node 3	1.155	47.896
Avg	1.155	47.894

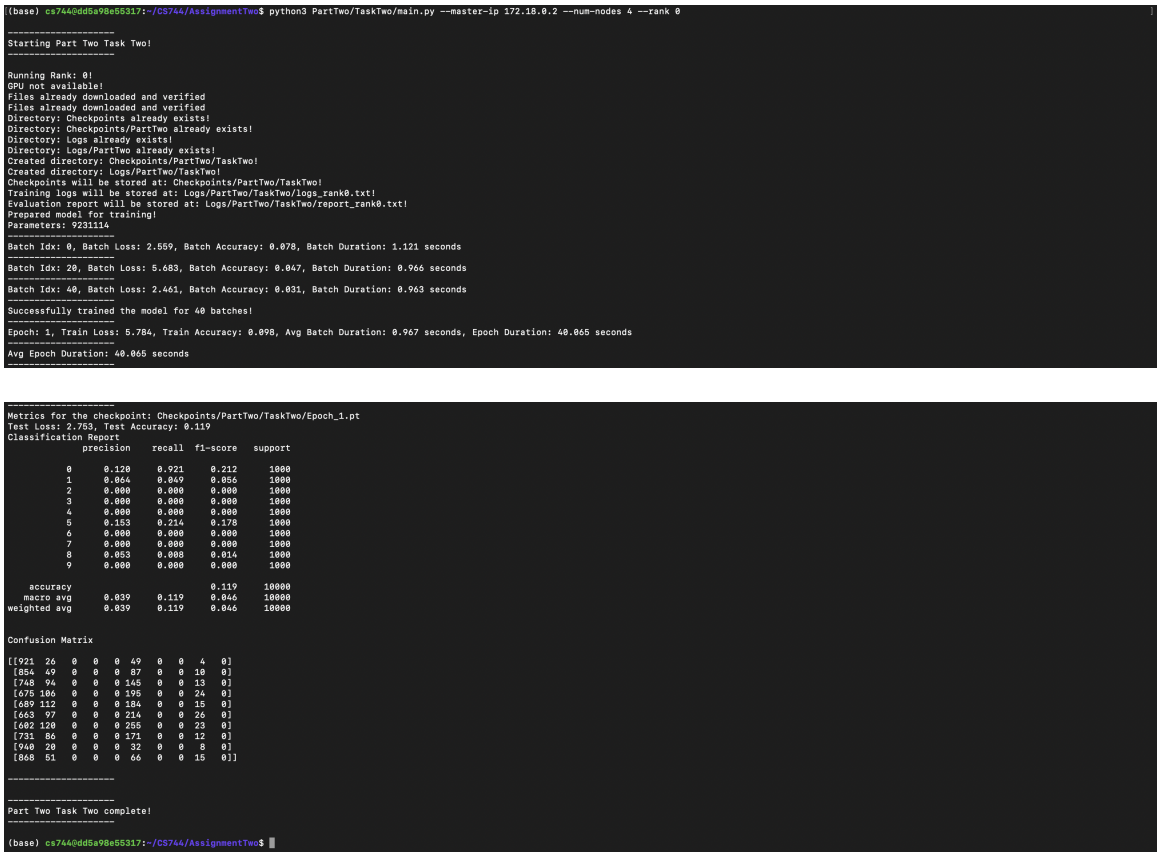

Figure 2: Output snapshots on Master Node for Part Two Task One.

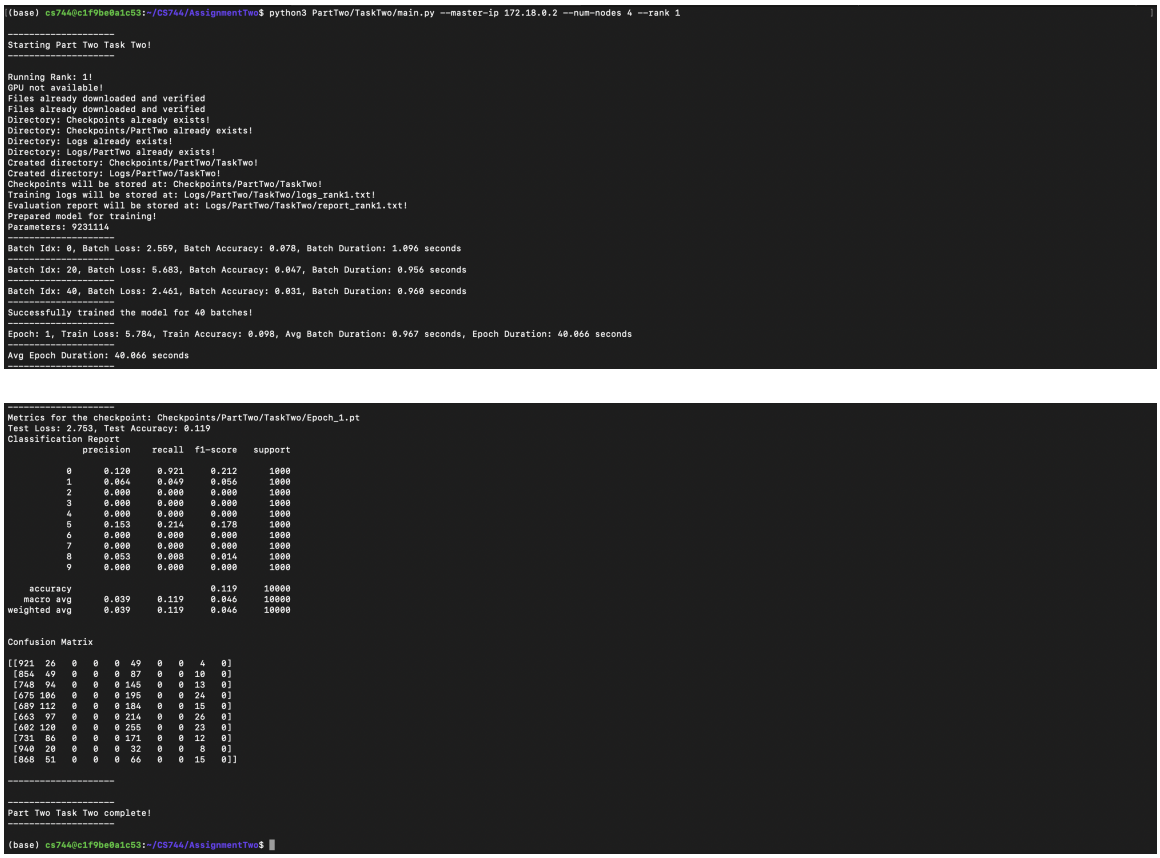

Figure 3: Output snapshots on Node 1 for Part Two Task One.

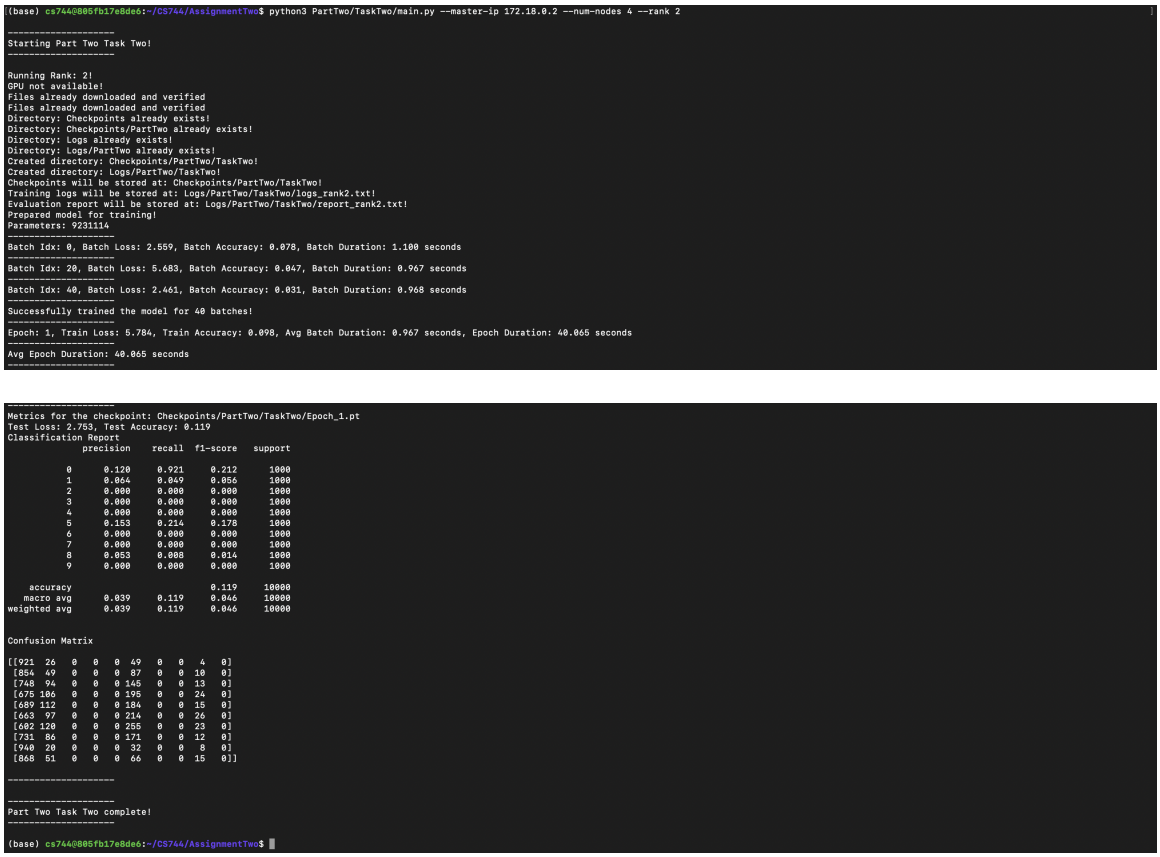

Figure 4: Output snapshots on Node 2 for Part Two Task One.

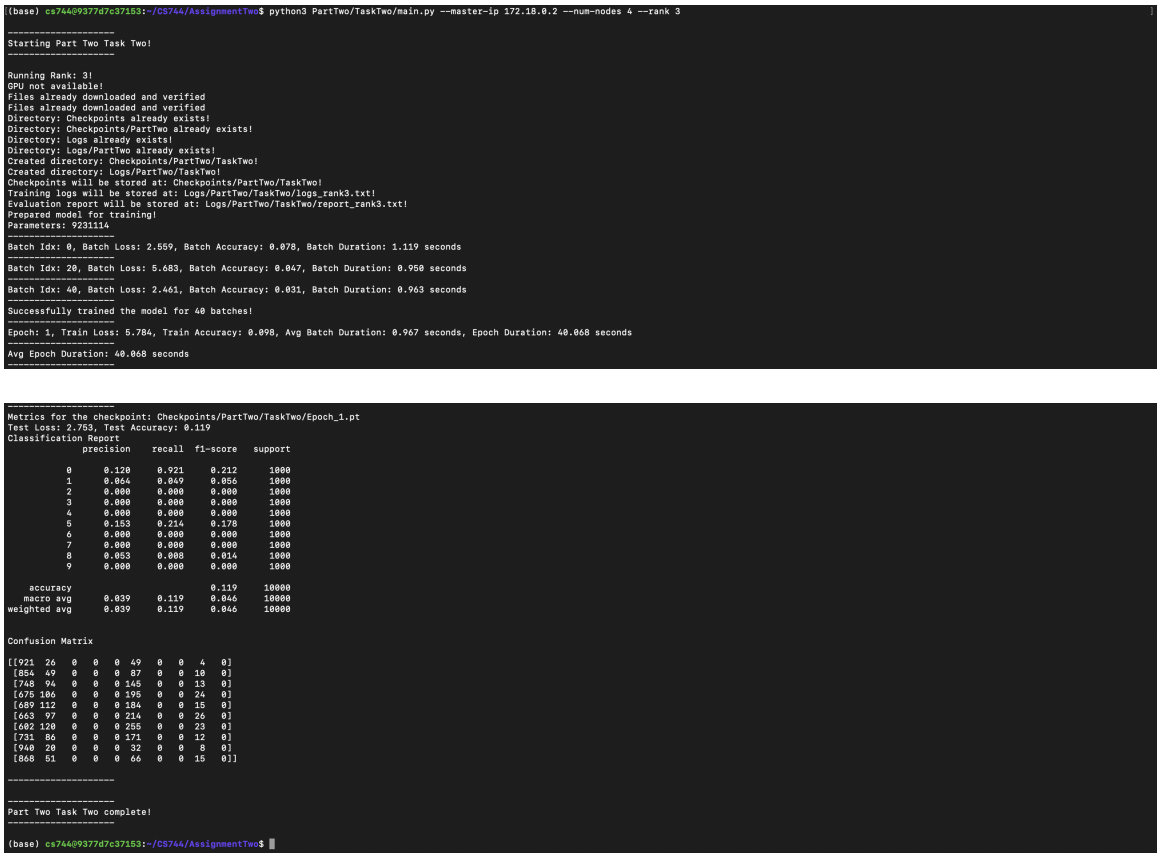

Figure 5: Output snapshots on Node 3 for Part Two Task One.

2.2 Part 2b: Sync gradient with allreduce using Gloo backend

2.2.1 Solution

The training process was completed successfully, producing the expected results. The final average train loss value was 5.784, the average test loss 2.753, while the test accuracy was 11.9%. Since we set the seed for PyTorch and NumPy to be 42 across the tasks, we were able to achieve the same loss and accuracy value for Task One and Task Two in Part Two. The expected difference in the batch duration is expected due to the difference in code. Table 3 denotes the average time it took to run 40 mini batch iterations and the epoch duration across the different nodes with a batch size of 64. From the table, we see that the average batch duration was 0.967 seconds with an average epoch duration of 40.066 seconds.

Table 2: Metrics across the different nodes for Part Two Task Two.

Node	**Avg Batch Duration**	**Epoch Duration**
Master Node	0.967	40.065
Node 1	0.967	40.066
Node 2	0.967	40.065
Node 3	0.967	40.068
Avg	0.967	40.066

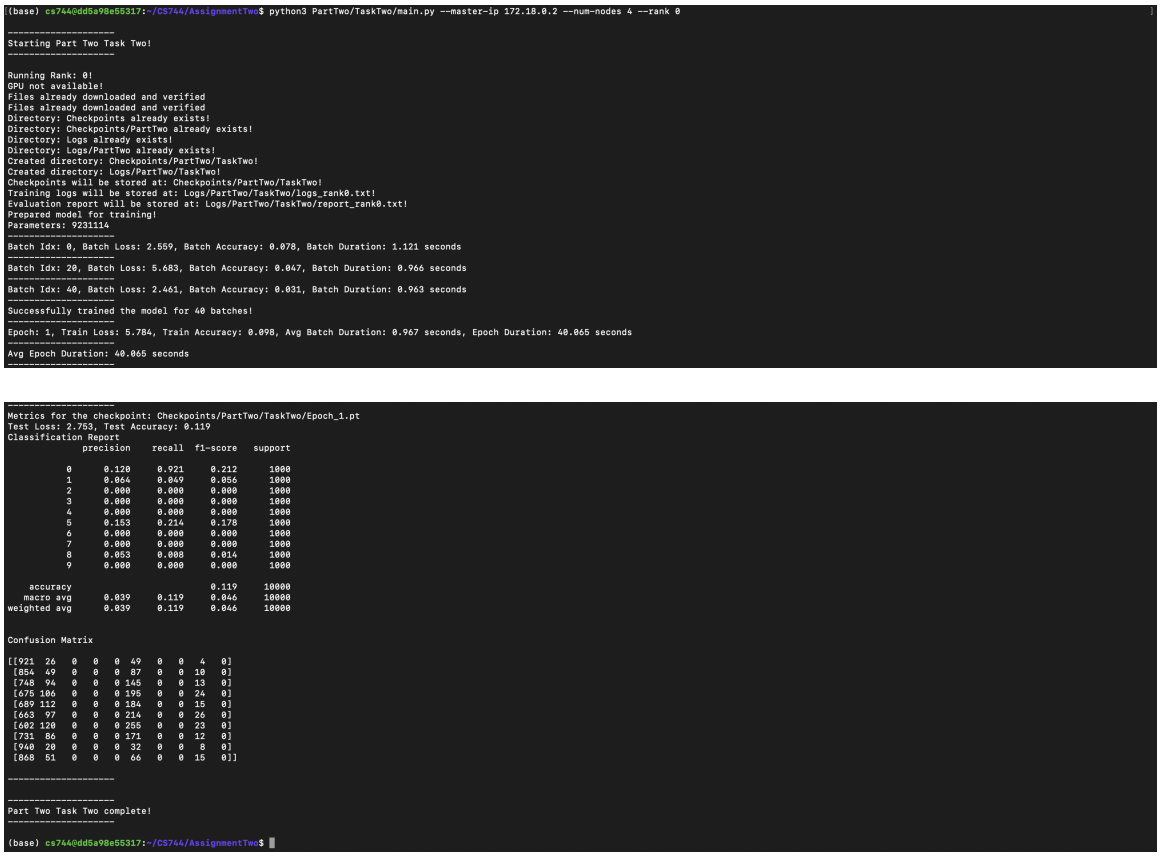

Figure 6: Output snapshots on Master Node for Part Two Task Two.

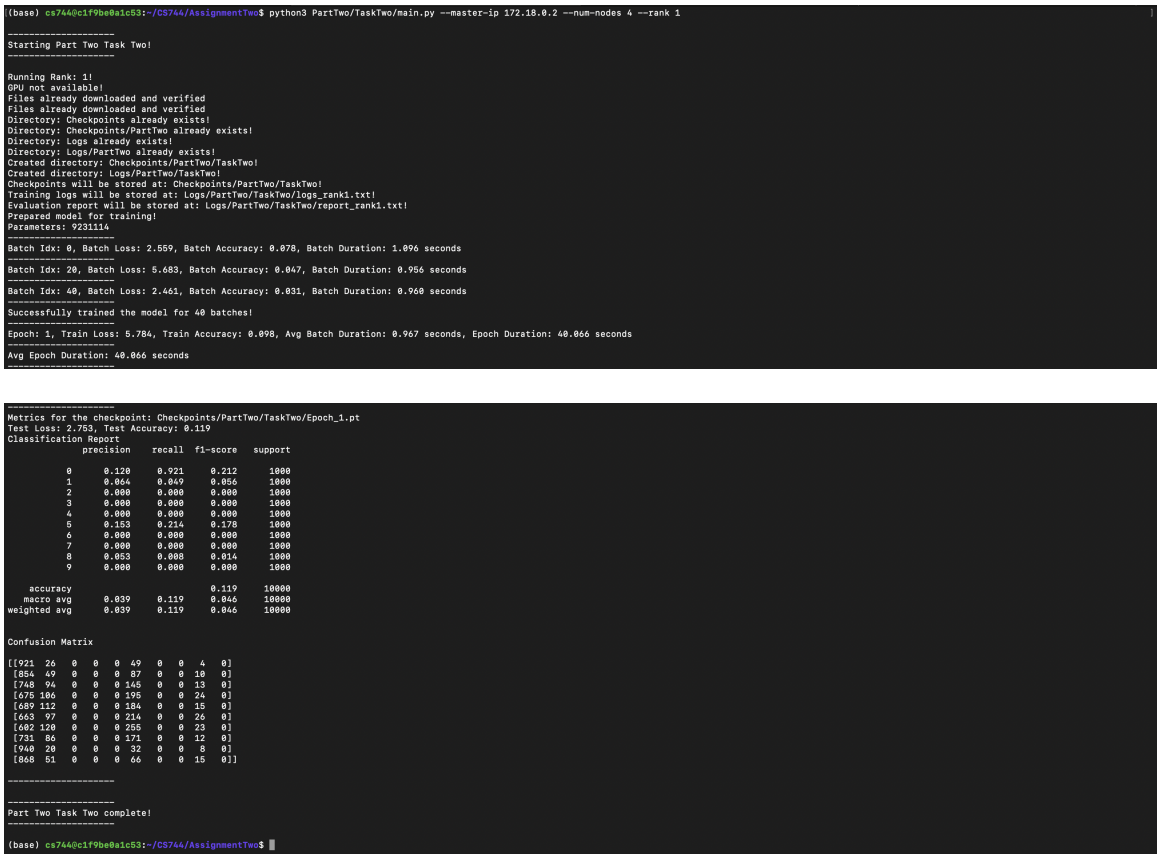

Figure 7: Output snapshots on Node 1 for Part Two Task Two.

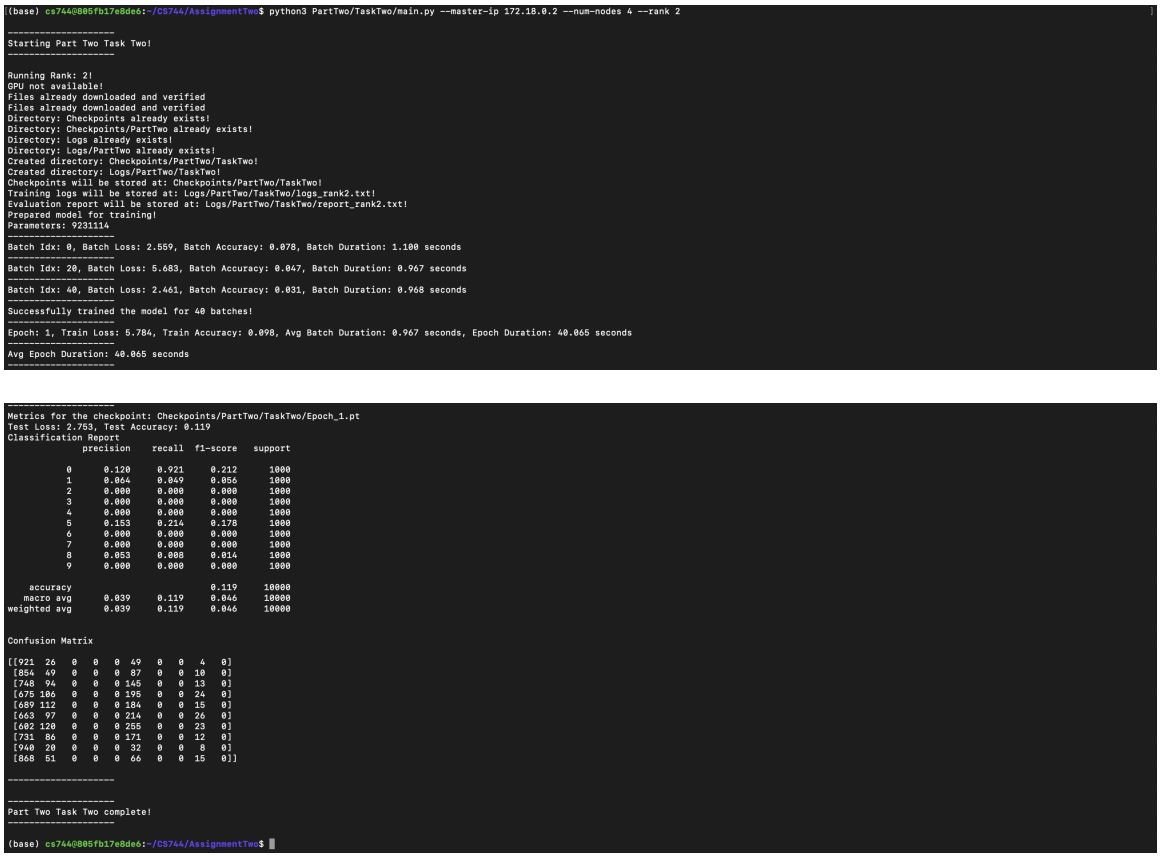

Figure 8: Output snapshots on Node 2 for Part Two Task Two.

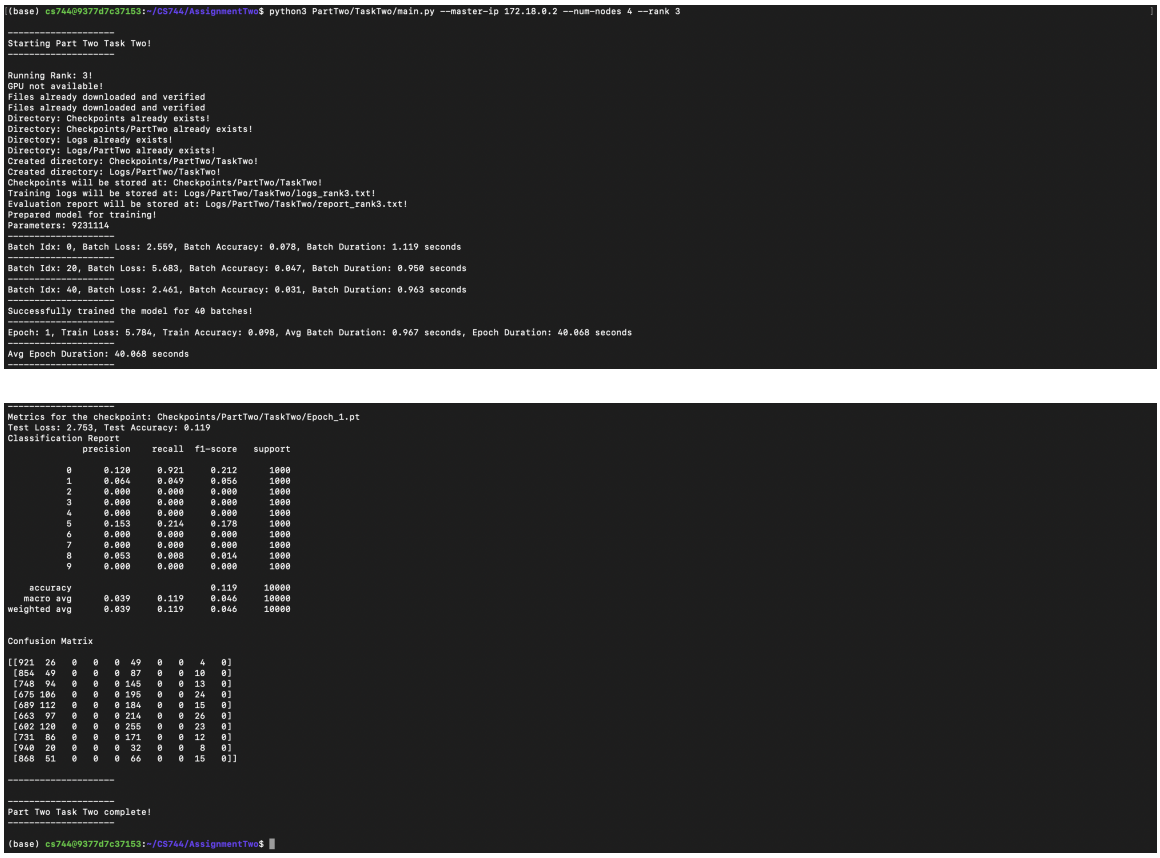

Figure 9: Output snapshots on Node 3 for Part Two Task Two.

3 Part Three: Distributed Data Parallel Training using Built in Module

3.1 Solution

The training process was completed successfully, producing the expected results. The final average train loss value was 5.784, the average test loss 2.753, while the test accuracy was 11.9%. Due to the same random seed for PyTorch and NumPy, we were able to achieve the same loss and accuracy value. We just made use of the inbuilt method instead of writing custom methods. Table **??** denotes the average time it took to run 40 mini batch iterations and the epoch duration across the different nodes with a batch size of 64. From the table, we see that the average batch duration was 0.943 seconds with an average epoch duration of 39.57 seconds.

Table 3: Metrics across the different nodes for Part Three.

Node	Avg Batch Duration	Epoch Duration
Master Node	0.943	39.066
Node 1	0.944	39.050
Node 2	0.943	39.060
Node 3	0.943	39.053
Avg	0.943	39.57

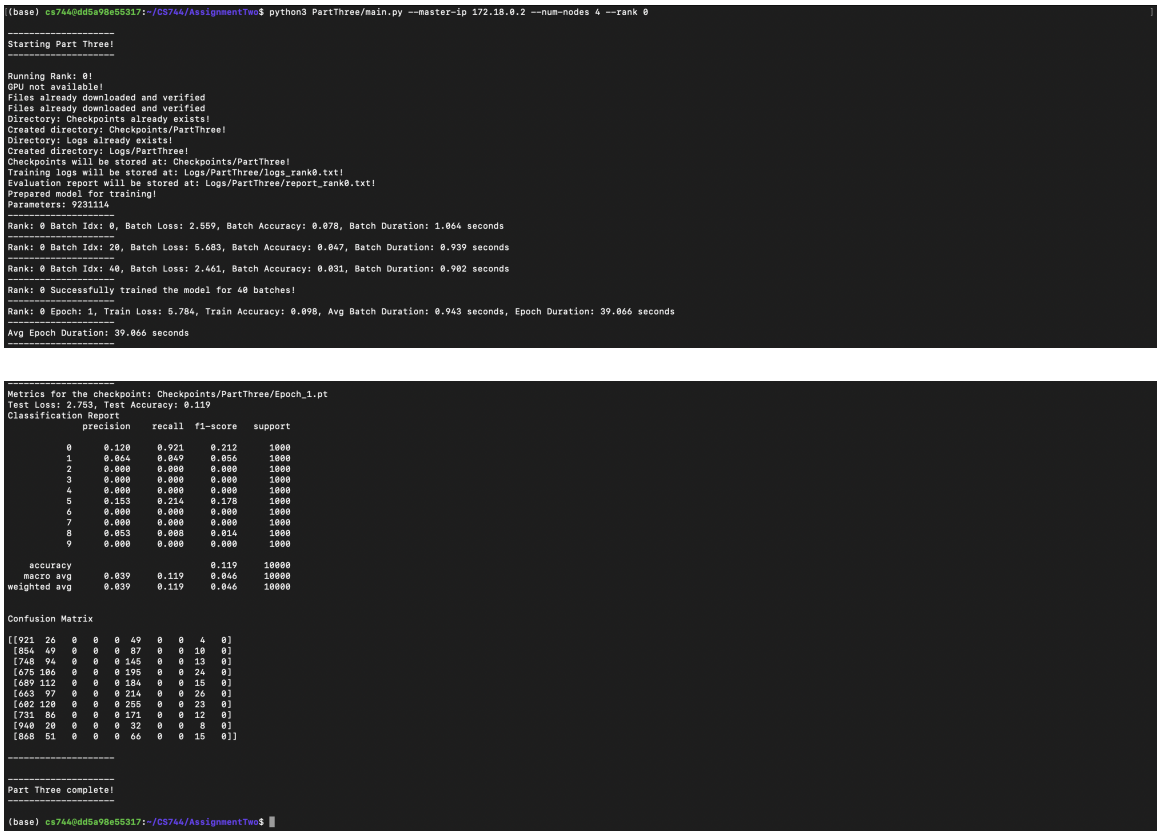

Figure 10: Output snapshots on Master Node for Part Three.

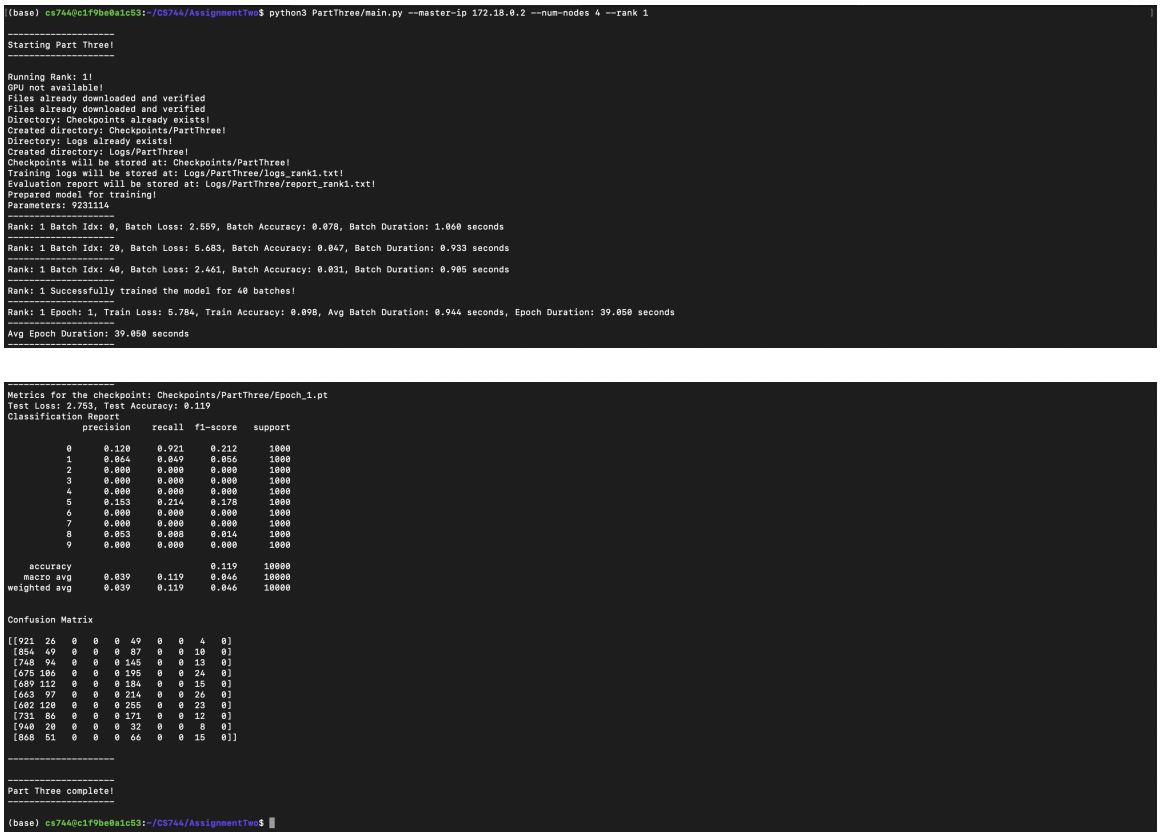

Figure 11: Output snapshots on Node 1 for Part Three.

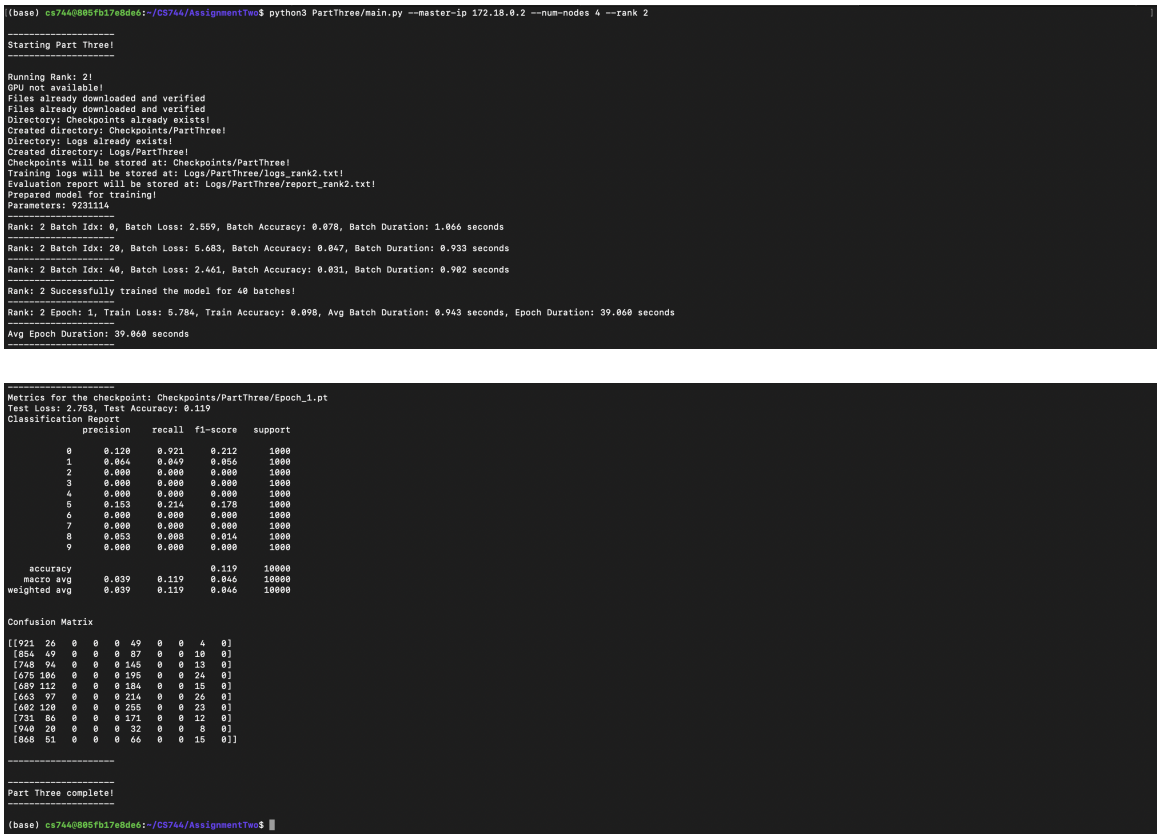

Figure 12: Output snapshots on Node 2 for Part Three.

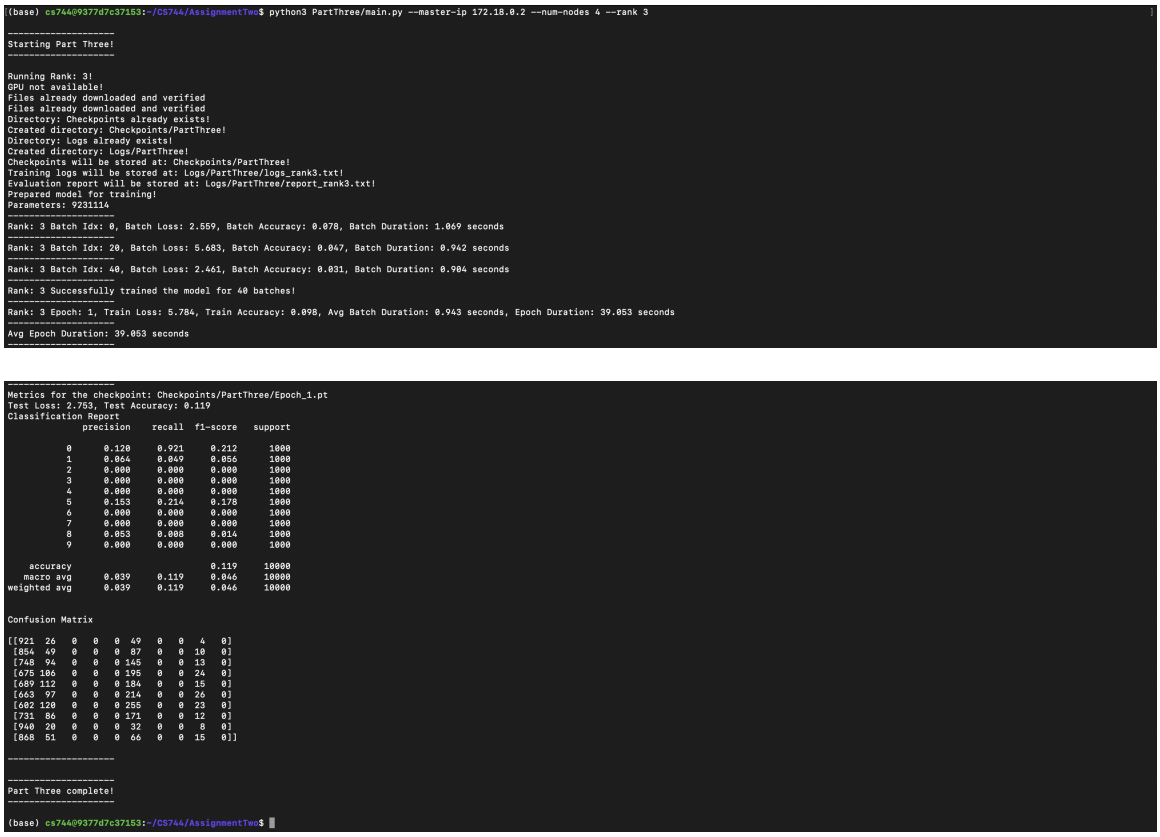

Figure 13: Output snapshots on Node 3 for Part Three.

4 Difference or lack of difference among different setups

The training time per iteration decreased as we progressed from Part 1 to DDP, with Part 1 taking the longest average time 1.477 seconds and DDP the shortest 0.943 seconds. This suggests that distributed training can significantly improve speed by parallelizing across nodes, compared to single-node training in Part 1.

Also, higher-level distributed APIs like AllReduce and DDP outperformed lower-level gather/scatter implementations. AllReduce uses a more efficient ring-based approach than scatter/gather. DDP further optimizes communication by bucketing small gradients before synchronizing, reducing total AllReduce calls. Leveraging multiple nodes and optimized distributed training APIs like

DDP can substantially reduce iteration time versus single-node training. The results highlight the performance benefits of distributed training, as well as efficient communication primitives like AllReduce.

In our analysis, we examined and compared the average loss across various setup methods. Overall, we observed that there is generally minimal variation in the average loss among all setups, indicating that different distributed configurations do not substantially affect training loss. This observation is logical because all setups use the same training data, maintain identical model structures, and adhere to consistent parameters such as total epochs, batch sizes, and training parameters. These factors contribute to the stability of average loss across setups.

However, there are still slight discrepancies in both the average loss and the loss recorded at intervals of every 20 iterations among different setups. This could be attributed to two potential factors: Firstly, the VGG models utilized in our study contain Batch Normalization (BN) layers, and the parameters within these layers are not synchronized across all nodes during distributed training. Consequently, this lack of synchronization introduces uncertainty and randomness into the training process, leading to marginal differences in loss across setups. Secondly, due to the inherent limitations of floating-point computation precision, minor errors may occur during training, further contributing to variability in loss computation results.

Similarly, while there is generally little disparity in test accuracy among different setups, slight variations still exist. These discrepancies likely stem from the same factors influencing average loss variation.

5 Comment on the scalability of distributed machine learning based on your results

We compare the average iteration time between the setups with different numbers of nodes to evaluate the scalability of the method. We can see that part 3 is 1.56 $\times$ faster than part 1. Thus, we can conclude that the speed does not improve linearly, even with an increase in the number of nodes.

Distributing training across multiple machines comes with a cost - it takes time to synchronize the gradient updates between nodes. How much overhead depends on the network connecting the machines. But when our data or model is too big for one machine, distributed data parallelism is our only option.

In our current setup with 4 Docker containers linked locally, the nodes communicate locally rather than external links. So the main overhead is just opening sockets and sending data back and forth. As the paper mentioned, we could

reduce this cost by doing a few local gradient updates before averaging across nodes.

Since our nodes are already well-connected on the same machine, the communication overhead isn't too bad. But as we scale up, optimizing the synchronization step will become more and more important. Tricks like local averaging before cross-node updates can help minimize the impact on training time. But distributing the work is the only way forward when limited by single-machine resources.

6 Contribution

- Badri
  1. Acted as the team lead and formulated the majority of the code.
  2. Wrote the code for the different helper functions, and also formulated the idea to use flags to train multiple variants of the model.
  3. Worked on training the VGG-16 model and generated results.
  4. Maintained version control with Git.
  5. Generated the report using LaTeX.
- Shreyas
  1. Formulated the code for Distributed Training.
  2. Implemented gather-scatter mechanism for gradient aggregation.
  3. Wrote the content for the report.
- Rohan
  1. Applied ring reduce to synchronize gradients across different nodes.
  2. Worked on code documentation.
  3. Formulated the code for the Distributed DataParallel module.
- Nevin
  1. Formulated the code for the Distributed DataParallel module.
  2. Worked on the content for the report.
  3. Documented the code

The entire team had an in-depth discussion about the results of our run.