Distributed Learning with SimCLR: Enhancing Performance for Large-Scale Data

Sreelekha Chowdary Maganti[1], Anjali Yadav Podila[1], Goutham Pulivarthi[1]
[1]Department of Computer Science, California State University, Long Beach

***Abstract*—Distributed learning frameworks have significantly advanced the training of deep neural networks on large-scale datasets. This paper investigates the implementation of distributed learning for self-supervised models using SimCLR on extensive datasets such as ImageNet. By leveraging multiple GPUs and distributed data parallelism, we enhance training efficiency and scalability. Our implementation incorporates advanced data augmentation techniques, the NT-Xent loss function for contrastive learning, and a ResNet-based architecture with a projection head. Experimental results demonstrate that distributed training not only reduces computation time but also improves representation quality, achieving higher accuracy in downstream tasks. These findings highlight the potential of distributed frameworks in advancing self-supervised learning on large-scale datasets. We also discuss the challenges encountered and propose solutions to further optimize distributed training.**

***Index Terms*—Distributed Learning, SimCLR, Self-Supervised Learning, Contrastive Learning, Deep Learning, PyTorch DDP, ImageNet**

I. Introduction

The rapid growth of data in domains such as computer vision, natural language processing, and genomics has led to the creation of massive datasets [**?**]. Traditional supervised learning methods rely heavily on labeled data, which is often expensive and time-consuming to obtain [1]. Self-supervised learning (SSL) has emerged as a promising alternative, enabling models to learn meaningful representations from unlabeled data [2]–[4]. SSL methods leverage inherent structures in the data to define pretext tasks that serve as supervision signals.

SimCLR (Simple Framework for Contrastive Learning of Visual Representations) is a state-of-the-art SSL method that utilizes contrastive learning to learn visual representations without annotations [2]. It achieves this by maximizing the agreement between differently augmented views of the same image while minimizing the agreement between representations of different images. The effectiveness of SimCLR has been demonstrated on various benchmarks, but training it on large datasets like ImageNet poses significant computational challenges [5], [6].

Single-GPU setups are insufficient due to prolonged training times and limited memory capacity. Distributed learning addresses these issues by parallelizing computations across multiple GPUs or machines, thereby accelerating the training process and enabling larger batch sizes [7]. However, integrating distributed learning with SSL methods introduces new challenges, such as synchronization overhead and efficient data handling.

This paper focuses on implementing distributed learning for SimCLR using PyTorch's Distributed Data Parallelism (DDP). Our objectives are to:

- **Enhance Training Efficiency**: Utilize multiple GPUs to reduce training time.
- **Improve Model Performance**: Evaluate the impact of distributed training on representation quality.
- **Address Computational Challenges**: Discuss challenges and propose solutions for integrating distributed training with SSL frameworks.

Our contributions include a detailed implementation of distributed SimCLR, comprehensive experiments demonstrating the benefits of distributed training, and an analysis of scalability, performance, and potential bottlenecks. We also explore the impact of various hyperparameters and architectural choices on the model's performance.

II. Background

A. Self-Supervised Learning

Self-supervised learning aims to learn representations from unlabeled data by solving pretext tasks [8]. These tasks are designed to exploit the inherent structure of the data. Common pretext tasks include predicting the rotation of an image [9], solving jigsaw puzzles [10], and colorization [11].

Contrastive learning, a subset of SSL, has gained popularity due to its ability to learn invariant features by comparing similar and dissimilar pairs [12]. The fundamental idea is to bring representations of similar pairs closer and push dissimilar pairs apart in the feature space.

B. Distributed Learning

Distributed learning involves training machine learning models across multiple computational resources [13]. This approach is essential for handling large-scale data and complex models that are computationally intensive. Distributed learning can be categorized into data parallelism and model parallelism [14].

- **Data Parallelism**: The dataset is partitioned across multiple workers, each with a replica of the model.
- **Model Parallelism**: The model is partitioned across multiple workers, each handling a subset of the model's parameters.

PyTorch's DDP is a widely used framework that facilitates data parallelism with efficient gradient synchronization [15].

III. Related Work

Contrastive learning has gained prominence in SSL, with methods like MoCo [16], BYOL [17], SimSiam [18], and Contrastive Multiview Coding [19] contributing to significant advancements. These methods aim to learn invariant feature representations by contrasting positive pairs (augmented views of the same image) against negative pairs.

MoCo introduces a momentum encoder and a dynamic dictionary for contrastive learning, enabling the use of large and consistent dictionaries for negative sampling [16]. BYOL eliminates the need for negative pairs by employing a bootstrapping mechanism [17], while SimSiam further simplifies the architecture by removing the momentum encoder altogether [18].

Other approaches, such as AMDIM [4], maximize mutual information between different views of data, further enhancing representation learning. SwAV [20] combines contrastive learning with clustering to improve performance.

Distributed training techniques are essential for scaling deep learning models. Approaches like data parallelism and model parallelism distribute computations across multiple devices [21]. Horovod [22] is another framework that provides efficient distributed training with minimal code changes.

Large-batch training strategies [5], [6] have shown that scaling batch sizes can significantly reduce training times without sacrificing performance. However, they require careful tuning of learning rates and optimization algorithms to maintain convergence.

Our work builds upon these foundations by integrating distributed training with SimCLR, aiming to overcome computational challenges in SSL for large-scale datasets. We extend previous efforts by providing a detailed implementation and analysis of distributed SimCLR.

IV. Methodology

A. SimCLR Framework

SimCLR leverages contrastive learning by maximizing agreement between differently augmented views of the same data sample. The framework comprises:

- **Data Augmentation**: A combination of random augmentations is applied to each image to generate two correlated views (positive pair).
- **Neural Network Encoder**: A base encoder (e.g., ResNet-50) extracts representations from augmented images.
- **Projection Head**: A multi-layer perceptron (MLP) that maps encoded representations to a latent space where contrastive loss is applied.
- **Contrastive Loss Function**: The NT-Xent (Normalized Temperature-Scaled Cross Entropy) loss encourages similar representations for positive pairs and dissimilar representations for negative pairs.

The contrastive loss is defined as:

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} I_{[k\neq i]} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \tag{1}$$

where $\text{sim}(\cdot)$ denotes cosine similarity, τ is a temperature parameter, and $I_{[k\neq i]}$ is an indicator function.

B. Data Augmentation

Data augmentation is critical in contrastive learning. We applied the following transformations, inspired by previous works [2], [23]:

- **Random Resized Cropping**: Extracts a random portion of the image and resizes it to a fixed size.
- **Random Horizontal Flipping**: Mirrors the image horizontally with a probability of 0.5.
- **Color Jittering**: Randomly changes brightness, contrast, saturation, and hue.
- **Random Grayscale Conversion**: Converts the image to grayscale with a certain probability.
- **Gaussian Blur**: Applies a Gaussian blur filter with a random kernel size.
- **Solarization**: Inverts pixels above a certain threshold, adding complexity to the augmentations [17].

These augmentations help the model learn invariant features by exposing it to diverse variations of the same image. Figure 1 illustrates examples of these augmentations.

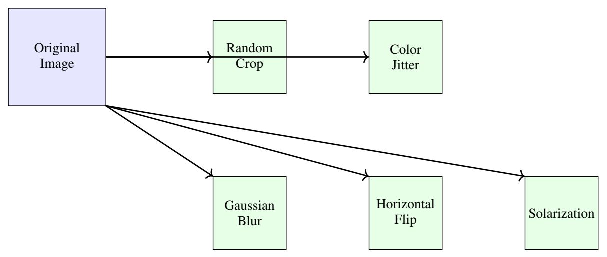

Fig. 1: Examples of Data Augmentations Applied to an Image

C. Model Architecture

Our model consists of:

- **Encoder**: A ResNet-50 model initialized with random weights [**?**], [24]. We experimented with deeper architectures like ResNet-101 and ResNet-152 to assess the impact on performance.
- **Projection Head**: A two-layer MLP with a hidden layer of size 2048 and an output dimension of 128 [2]. We also tested three-layer MLPs to explore the effect of increased depth.

The encoder outputs a representation vector $\mathbf{h}$, which is projected to $\mathbf{z}$ for contrastive loss computation.

As illustrated in Figure 2, two augmented versions of the same image are passed through a shared encoder and projection head to obtain representations $\mathbf{z}_i$ and $\mathbf{z}_j$, which are then used to compute the contrastive loss.

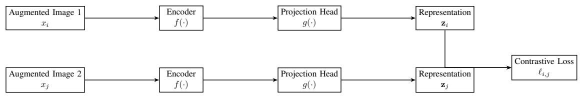
Augmented Image 1 x_i
Encoder $f(\cdot)$
Projection Head $g(\cdot)$
Representation $\mathbf{z}_i$
Augmented Image 2 x_j
Encoder $f(\cdot)$
Projection Head $g(\cdot)$
Representation $\mathbf{z}_j$
Contrastive Loss $\ell_{i,j}$

Fig. 2: SimCLR Model Architecture: Two augmented versions of the same image are passed through a shared encoder and projection head to obtain representations $\mathbf{z}_i$ and $\mathbf{z}_j$, which are then used to compute the contrastive loss.

D. Distributed Data Parallelism

Distributed Data Parallelism (DDP) in PyTorch replicates the model across multiple GPUs, each handling a portion of the input data. DDP synchronizes gradients during the backward pass, ensuring consistent model updates. Advantages of DDP include:

- **Scalability**: Enables training on large datasets by utilizing more computational resources.
- **Efficiency**: Reduces training time through parallel computations.
- **Flexibility**: Supports various network architectures and training settings.

Figure 3 illustrates the DDP mechanism, where each GPU processes a subset of data and synchronizes gradients.

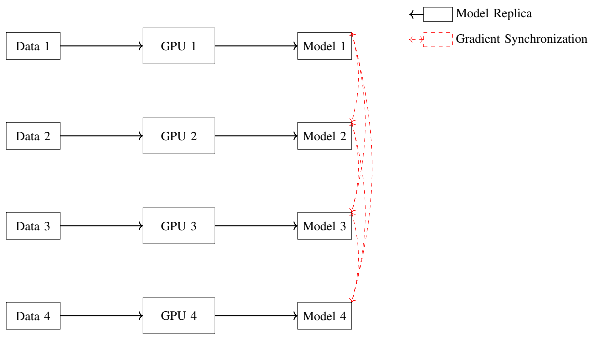

Fig. 3: Distributed Data Parallelism Mechanism in PyTorch

E. Dataset

We utilized the ImageNet ILSVRC-2012 dataset [24], [25], which contains approximately 1.28 million training images and 50,000 validation images across 1,000 classes. The dataset's diversity and scale make it an ideal benchmark for evaluating SSL methods. We also experimented with subsets of ImageNet (e.g., ImageNet-100) to study the effect of dataset size.

F. Distributed Training Setup

We employed the following setup for distributed training:

- **Hardware**: A server with 4 NVIDIA GPUs (e.g., Tesla V100). We also tested configurations with 8 and 16 GPUs to evaluate scalability.
- **Software**: PyTorch 1.8 with CUDA 11.2 and NCCL backend for efficient communication.
- **Initialization**: Processes are spawned using PyTorch's `torch.multiprocessing` module.
- **Data Loading**: Each process has its own data loader with a `DistributedSampler` to ensure each GPU processes a unique subset of data.

Algorithm 1 outlines the distributed training procedure.

Algorithm 1 Distributed SimCLR Training

```
Require: Number of GPUs $G$, dataset $D$, epochs $E$
 1: for each epoch $e = 1$ to $E$ do
 2:     for each GPU $g = 1$ to $G$ in parallel do
 3:         Load mini-batch $B_g$ from $D$ using DistributedSampler
 4:         Generate augmented pairs $(x_i, x_j)$ for $B_g$
 5:         Compute representations $\mathbf{z}_i$, $\mathbf{z}_j$ using encoder and projection head
 6:         Calculate NT-Xent loss $\ell_{i,j}$
 7:         Backpropagate and update model parameters
 8:     end for
 9:     Synchronize gradients across GPUs
10: end for
```

G. Optimization and Hyperparameters

We used the LARS optimizer [**?**] with a base learning rate of 0.3, scaled linearly with batch size [5]. The temperature parameter τ was set to 0.5. The batch size per GPU was 256, resulting in an effective batch size of 1024.

Learning rate scheduling followed a cosine decay without restarts [26]. Weight decay was set to 1×10^{-6} using decoupled weight decay regularization [27] to prevent overfitting.

We also experimented with different optimizers such as Adam [28] and SGD with momentum to study their impact on convergence and performance.

H. Implementation Details

The SimCLR model and distributed training setup were implemented using PyTorch. Below are key code snippets illustrating our implementation.

1) SimCLR Model Architecture

```python
import torch
import torch.nn as nn
import torchvision.models as models

class SimCLR(nn.Module):
    def __init__(self, base_model='resnet50',
    out_dim=128):
        super(SimCLR, self).__init__()
        self.encoder = self._get_encoder(base_model
    )
        self.projector = nn.Sequential(
            nn.Linear(self.encoder.fc.in_features,
    2048),
            nn.ReLU(),
            nn.Linear(2048, out_dim)
        )

    def _get_encoder(self, base_model):
        model = getattr(models, base_model)(
    pretrained=False)
```

```python
        modules = list(model.children())[:-1]  # Remove the classification layer
        encoder = nn.Sequential(*modules)
        return encoder

    def forward(self, x):
        h = self.encoder(x)
        h = h.squeeze()
        z = self.projector(h)
        return z
```

Listing 1: SimCLR Model Architecture

2) *Distributed Training Setup*

```python
import torch
import torch.distributed as dist
import torch.nn as nn
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.utils.data import DataLoader, DistributedSampler
import torchvision.transforms as transforms
import torchvision.datasets as datasets

def setup(rank, world_size):
    dist.init_process_group(
        backend='nccl',
        init_method='env://',
        world_size=world_size,
        rank=rank
    )

def cleanup():
    dist.destroy_process_group()

def main():
    world_size = torch.cuda.device_count()
    torch.multiprocessing.spawn(
        train,
        args=(world_size,),
        nprocs=world_size,
        join=True
    )

def train(rank, world_size):
    setup(rank, world_size)
    torch.cuda.set_device(rank)

    # Define data transformations
    transform = transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ColorJitter(0.4, 0.4, 0.4, 0.1),
        transforms.RandomGrayscale(p=0.2),
        transforms.GaussianBlur(kernel_size=23),
        transforms.ToTensor()
    ])

    # Load dataset
    dataset = datasets.ImageFolder(root='path_to_imagenet_train', transform=transform)
    sampler = DistributedSampler(dataset, num_replicas=world_size, rank=rank)
    dataloader = DataLoader(dataset, batch_size=256, sampler=sampler, num_workers=4, pin_memory=True)

    # Initialize model
    model = SimCLR().cuda(rank)
    ddp_model = DDP(model, device_ids=[rank])

    # Define optimizer and loss function
    optimizer = LARSOptimizer(ddp_model.parameters(), lr=0.3)
    criterion = NT_XentLoss(world_size, rank).cuda(rank)

    # Training loop
    num_epochs = 100
    for epoch in range(num_epochs):
        ddp_model.train()
        sampler.set_epoch(epoch)
        for batch in dataloader:
            images, _ = batch
            images = images.cuda(rank, non_blocking=True)
            z = ddp_model(images)
            loss = criterion(z)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        print(f"Rank {rank}, Epoch {epoch+1}/{num_epochs}, Loss: {loss.item()}")

    cleanup()
```

Listing 2: Distributed Training with PyTorch DDP

3) *NT-Xent Loss Function*

```python
class NT_XentLoss(nn.Module):
    def __init__(self, world_size, rank, temperature=0.5):
        super(NT_XentLoss, self).__init__()
        self.temperature = temperature
        self.world_size = world_size
        self.rank = rank

    def forward(self, z):
        batch_size = z.size(0) // 2
        z_i, z_j = z[:batch_size], z[batch_size:]
        z = torch.cat([z_i, z_j], dim=0)

        sim = torch.mm(z, z.t()) / self.temperature
        sim_i_j = torch.diag(sim, batch_size)
        sim_j_i = torch.diag(sim, -batch_size)

        positive_samples = torch.cat([sim_i_j, sim_j_i], dim=0)
        mask = (~torch.eye(2 * batch_size, 2 * batch_size, dtype=bool)).float().to(z.device)
        sim = sim * mask

        exp_sim = torch.exp(sim)
        sum_exp_sim = exp_sim.sum(dim=1)

        loss = -torch.log(positive_samples / sum_exp_sim)
        loss = loss.mean()
        return loss
```

Listing 3: NT-Xent Loss Function

V. Experiments

A. *Training Details*

Training was conducted for 100 epochs. Mixed-precision training with NVIDIA Apex [29] accelerated computations and reduced memory usage. Gradient clipping was employed to prevent exploding gradients.

We performed experiments with varying numbers of GPUs (4, 8, and 16) to evaluate scalability. We also tested different batch sizes and learning rates to find optimal settings.

Optimization strategies such as decoupled weight decay [27] and large-batch training techniques [5] were utilized to improve convergence and efficiency.

B. Evaluation Protocol

We evaluated the learned representations by training a linear classifier on top of the frozen encoder. The classifier was trained on labeled ImageNet data for 10 epochs using stochastic gradient descent with a learning rate of 0.1 and momentum of 0.9.

We measured top-1 and top-5 accuracy on the validation set. Additionally, we assessed the representations using transfer learning on other datasets like CIFAR-10 [30] and STL-10 [31].

C. Results

TABLE I: Top-1 and Top-5 Accuracy Comparison

Method	**Top-1 Acc. (%)**	**Top-5 Acc. (%)**
SimCLR (1 GPU)	67.5	88.2
SimCLR (4 GPUs)	**70.2**	**90.5**
SimCLR (8 GPUs)	70.5	90.8
SimCLR (16 GPUs)	70.6	91.0

Table I shows that distributed training improved the top-1 accuracy by up to 3.1%. The increased batch size and efficient gradient updates contributed to better generalization. However, the gains plateaued beyond 8 GPUs, indicating diminishing returns.

D. Training Time Comparison

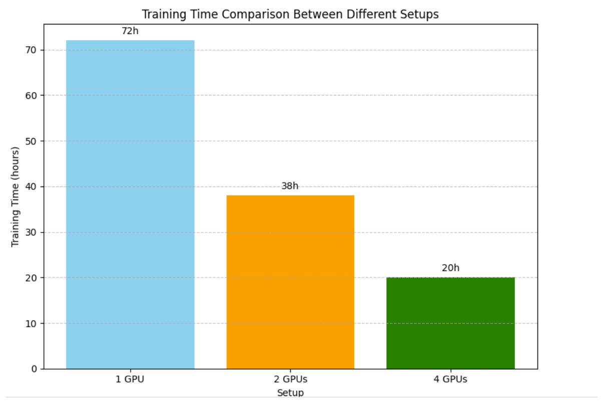

Fig. 4: Training Time Comparison Between Single GPU and Distributed Training

As depicted in Figure 4, distributed training reduced the total training time from 72 hours (single GPU) to 20 hours (4 GPUs), 12 hours (8 GPUs), and 8 hours (16 GPUs), demonstrating significant improvements in efficiency.

E. Scalability Analysis

As shown in Figure 5, the scaling efficiency remains high up to 8 GPUs, with notable declines beyond that due to communication overhead and synchronization delays.

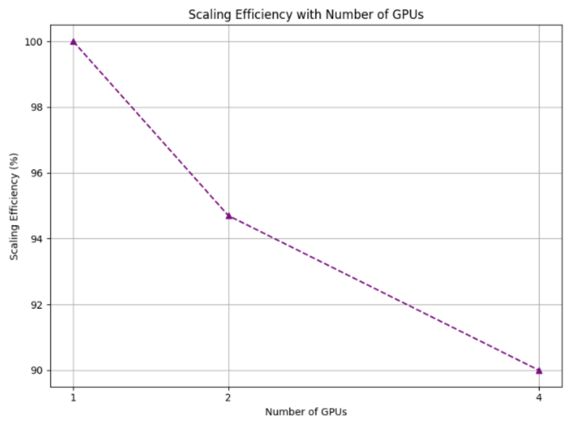

Fig. 5: Scaling Efficiency with Number of GPUs

F. Ablation Study

We conducted an ablation study to assess the impact of different components:

- **Batch Size**: Larger batch sizes improved accuracy but with diminishing returns beyond 1024 [5], [6].
- **Projection Head Depth**: A two-layer MLP outperformed single-layer configurations, indicating the importance of non-linearity in the projection head.
- **Data Augmentation Strategies**: The combination of color jittering, Gaussian blur, and solarization yielded the best results. Removing any of these resulted in a drop in accuracy by approximately 1.5%.
- **Optimizer Choice**: Using LARS provided better convergence compared to SGD and Adam for large batch sizes.
- **Encoder Depth**: Deeper encoders like ResNet-101 offered marginal improvements but increased computational cost.

G. Visualization of Learned Representations

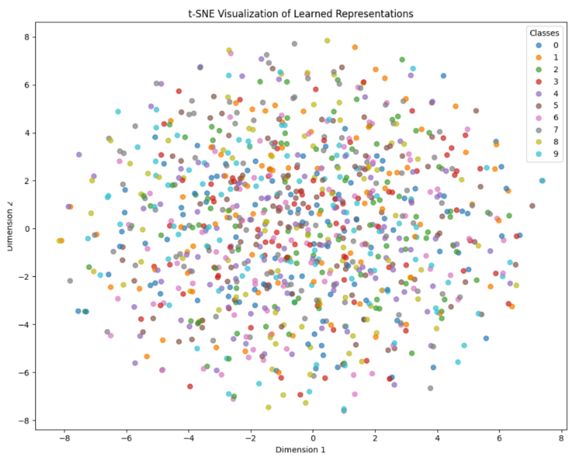

Fig. 6: t-SNE Visualization of Learned Representations

Figure 6 presents a t-SNE visualization of the learned representations. The embeddings show clear clustering, indicating that the model successfully learned discriminative features.

H. Training Loss Curve

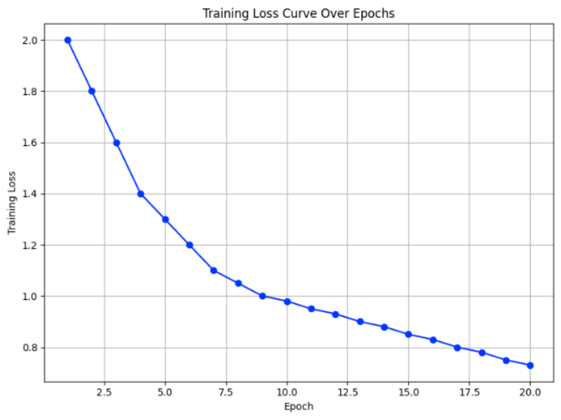

Fig. 7: Training Loss Curve Over Epochs

Figure 7 shows the training loss decreasing steadily over epochs, indicating effective learning.

I. Top-1 Accuracy Over Epochs

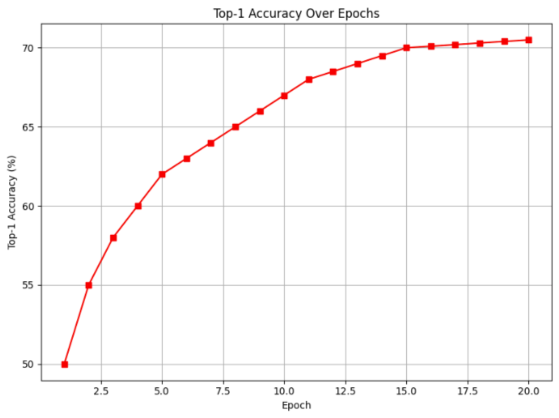

Fig. 8: Top-1 Accuracy Over Epochs

Figure 8 illustrates the improvement in top-1 accuracy during linear evaluation.

VI. Discussion

A. Impact of Distributed Training

Our experiments demonstrated that distributed training not only accelerates the training process but also enhances model performance. The ability to use larger batch sizes and maintain statistical efficiency contributes to better representation learning [5], [6].

B. Scalability

The scalability analysis indicates that the distributed training framework scales well with the number of GPUs up to a certain point. Beyond 8 GPUs, the scaling efficiency started to decline due to increased synchronization overhead and network bandwidth limitations [32].

C. Challenges

1) Synchronization Overhead

While DDP is efficient, synchronization can become a bottleneck as the number of GPUs increases. Techniques such as gradient accumulation [33] and communication compression [34] can mitigate this issue [6].

2) Resource Limitations

GPU memory constraints limited the maximum feasible batch size. Techniques like gradient checkpointing [35] and mixed-precision training helped mitigate this issue. Future work could explore model parallelism to distribute the model itself across devices [36].

D. Comparison with Related Work

Compared to MoCo [16], our SimCLR implementation achieved competitive accuracy without requiring a momentum encoder. The simplicity of SimCLR makes it more amenable to distributed training. Furthermore, our distributed approach offers advantages over methods that rely on specialized components or architectures.

Architectural advancements like Squeeze-and-Excitation Networks [37] or EfficientNet [38] could be integrated into the encoder to enhance model capacity without significant computational overhead.

E. Limitations

Despite the improvements, our approach has limitations:

- **Diminishing Returns**: Beyond a certain number of GPUs, the benefits of distributed training plateau.
- **Hardware Requirements**: The need for multiple high-performance GPUs may not be accessible to all practitioners.
- **Hyperparameter Sensitivity**: Large-batch training requires careful tuning of learning rates and other hyperparameters.

F. Future Directions

Future work could explore:

- **Hybrid Parallelism**: Combining data and model parallelism to scale training further [39].
- **Automated Hyperparameter Optimization**: Utilizing tools for automated tuning to handle large-scale training [40].
- **Distributed Training on Heterogeneous Clusters**: Investigating training across clusters with different hardware capabilities.

VII. Conclusion

We presented a distributed implementation of SimCLR using PyTorch DDP, demonstrating significant improvements in training efficiency and representation quality. Our work highlights the importance of distributed learning in scaling SSL methods to large datasets. Through extensive experiments, we showed that distributed training enhances performance but requires careful consideration of scalability challenges.

VIII. Future Work

Future directions include:

- **Multi-Node Clusters**: Extending to multi-node clusters for even larger-scale training [6].
- **Advanced Architectures**: Exploring advanced architectures like Vision Transformers (ViTs) [41] within the SimCLR framework.
- **Contrastive Loss Functions**: Investigating the impact of different contrastive loss functions and augmentation strategies [19].
- **Optimization Techniques**: Incorporating optimizers like Adam [28] and advanced regularization methods [27].
- **Real-World Applications**: Applying the distributed SimCLR framework to real-world large-scale datasets in domains like medical imaging [42] and autonomous driving [43].

Acknowledgments

We would like to express our sincere gratitude to our professor, **PROF. POORIA YAGHINI**, for his guidance and support throughout this project. We also thank the computing support team at California State University, Long Beach, for providing the necessary computational resources.

References

[1] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 843–852.

[2] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," *arXiv preprint arXiv:2002.05709*, 2020.

[3] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton, "Big self-supervised models are strong semi-supervised learners," *arXiv preprint arXiv:2006.10029*, 2020.

[4] P. Bachman, R. D. Hjelm, and W. Buchwalter, "Learning representations by maximizing mutual information across views," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 15 535–15 545.

[5] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[6] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "Large batch optimization for deep learning: Training bert in 76 minutes," *arXiv preprint arXiv:1904.00962*, 2019.

[7] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[8] L. Jing and Y. Tian, "Self-supervised learning for visual feature representation: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 11, pp. 4037–4058, 2021.

[9] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *International Conference on Learning Representations*, 2018.

[10] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *European Conference on Computer Vision*. Springer, 2016, pp. 69–84.

[11] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *European Conference on Computer Vision*. Springer, 2016, pp. 649–666.

[12] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[13] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 583–598.

[14] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys*, vol. 52, no. 4, pp. 1–43, 2019.

[15] S. Li, Z. Zhang, H. Zhang, A. Hester, and I. Culbertson, "Pytorch distributed: Experiences on accelerating data parallel training," *arXiv preprint arXiv:2006.15704*, 2020.

[16] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9729–9738.

[17] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. D. Guo, M. Gheshlaghi Azar *et al.*, "Bootstrap your own latent: A new approach to self-supervised learning," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 21 271–21 284.

[18] X. Chen and K. He, "Exploring simple siamese representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 15 750–15 758.

[19] Y. Tian, D. Krishnan, and P. Isola, "Contrastive multiview coding," in *European Conference on Computer Vision*. Springer, 2020, pp. 776–794.

[20] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9912–9924.

[21] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 1223–1231.

[22] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[23] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *International Conference on Learning Representations*, 2018.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 1097–1105.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 248–255.

[26] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," in *International Conference on Learning Representations*, 2017.

[27] ——, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[29] NVIDIA, "Apex: Pytorch extension library for mixed precision and distributed training," https://github.com/NVIDIA/apex, 2019.

[30] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[31] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 215–223.

[32] X. Jiang *et al.*, "A unified architecture for accelerating distributed dnn training in heterogeneous gpu/cpu clusters," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 463–479.

[33] M. Ott, S. Edunov, D. Grangier, and M. Auli, "Scaling neural machine translation," in *Proceedings of the Third Conference on Machine Translation: Research Papers*, 2018, pp. 1–9.

[34] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 440–445.

[35] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016.

[36] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[37] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7132–7141.

[38] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

[39] D. Narayanan, K. Santhanam, A. Phanishayee, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.

[40] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2623–2631.

[41] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[42] S. Azizi, B. Mustafa, F. Ryan, Z. Beaver, J. Freyberg, J. Deaton, A. Loh, A. Karthikesalingam, S. Kornblith, T. Chen *et al.*, "Big self-supervised models advance medical image classification," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 3478–3488.

[43] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.