Autoencoder-Based Gradient Compression for Distributed Training

Lusine Abrahamyan, Yiming Chen, Giannis Bekoulis, Nikos Deligiannis
Department of Electronics and Informatics, Vrije Universiteit Brussel, Brussels, Belgium
imec, Kapeldreef 75, B- 3001, Leuven, Belgium
Email: {alusine, ychen, gbekouli, ndeligia}@etrovub.be

***Abstract*—Large-scale distributed training has recently been proposed as a solution to speed-up the training of deep neural networks on huge datasets. Distributed training, however, entails high communication rates for gradient exchange among computing nodes and requires expensive high-bandwidth network infrastructure. Various gradient compression methods have been proposed to overcome this limitation, including sparsification, quantization, and entropy encoding of the gradients. However, most existing methods leverage only the *intra-node* information redundancy, that is, they compress gradients at each node *independently*. In contrast, we advocate that the gradients across the nodes are correlated and propose a method to leverage this *inter-node* redundancy to obtain higher compression rates. In this work, we propose the Learned Gradient Compression (LGC) framework to reduce communication rates within a distributed training with the parameter server communication protocol. Our framework leverages an autoencoder to capture the common information in the gradients of the distributed nodes and eliminate the transmission of redundant information. Our experiments show that the proposed approach achieves significantly higher gradient compression ratios than state-of-the-art approaches like DGC and ScaleCom.**

***Index Terms*—Deep learning, data-parallel distributed training, gradient compression, autoencoders.**

I. Introduction

In recent years, the progress in the field of deep learning has been achieved by training models with a large number of parameters on a huge amount of data. With the growth in the model parameters and the dataset size, the training of such models using a single machine becomes very expensive in terms of latency. One solution to this problem can be training in a number of computing nodes in parallel, a.k.a., distributed training [1]. One of the most used methods is data-parallel distributed training [2], where each node has a replica (i.e., a copy) of the model and access to a chunk of the data. In data-parallel distributed training, due to the transmission of gradients from one node to the other, there can be a communication and latency overhead. The size of these gradients can reach hundreds of megabytes (MBs) per iteration. For example, the size of the gradient tensor of the ResNet101 [3] is 170MB. This paper focuses on data-parallel distributed training and proposes a new framework to reduce the overhead caused by the transfer of the gradients.

Various methods have been proposed to tackle the problem of communication overhead caused by the transmission of large gradients. These methods include gradient sparsification [4]–[6], quantization [6], [7] and entropy coding [8]. Gradient sparsification methods transfer a portion of the gradient (instead of the full gradient) depending on some importance metric. Deep Gradient Compression (DGC) [5], for example, applied top-k gradient selection to obtain sparse gradients and achieved up to 99.9% gradient sparsification without loss in accuracy. Alternatively, gradient quantization approaches perform quantization of the gradients prior to transmission. The authors of [7], for instance, trained the ResNet-152 [3] network, with the 8-bit quantized gradients, to full accuracy on ImageNet [9] $1.8\times$ faster than the version with full-precision gradients.

Despite their effectiveness, these methods explore only the intra-node gradient redundancy. In this work, we propose to leverage the correlation between the gradients of the different nodes to achieve further compression gains. An attempt to explore the redundancies of gradients of different nodes was also made in [8], [10]. The method described in [8] is based on distributed source coding, realized by low-density parity-check (LDPC) codes, which leads to an impractically high decoding latency and complexity. In contrast, our method employs a lightweight autoencoder to compress the gradients, which considerably reduces the communication rate without compromising the encoding and decoding speed. In [10], the authors also utilized the correlation between the gradient tensors. They showed that the cosine distance between the gradient residuals at the different nodes decreases fast over the iterations. Based on this similarity, the authors of [10] proposed ScaleCom, an approach using a new Cyclic Local Top-k (CLT-k) compressor for the ring-allreduce protocol. Within their compression scheme, one of the workers performs top-k selection and all other workers take the values at the indices selected by the leading worker. Compared with this approach, our Learned Gradient Compression (LGC) framework can provide higher compression ratios because of the introduction of an autoencoder-based distributed compression approach.

In summary, the contributions of this work are:

- We propose a novel autoencoder-based framework for performing distributed gradient compression by leveraging the correlation between them. To the best of our knowledge, this is the first attempt to use autoencoders to capture the correlation across gradients, in order to compress them.
- We study experimentally the statistical dependency among gradients of different nodes using information-theoretic metrics and show that there is a significant rate reduction

that can be achieved if these correlations are exploited.

- We experimentally evaluate our method against different benchmarks—including uncompressed gradients (baseline method), and the state-of-the-art DGC [5] and ScaleCom [10] methods—and show that our framework can achieve significant rate reductions for different tasks, models and datasets.

The remainder of this paper is structured as follows. Section II describes our autoencoder to leverage the correlation between gradients and Section III presents how the autoencoder can be used within the proposed LGC framework for distributed training. Section IV elaborates on the experimental validation of the approach and Section V draws the conclusion.

II. Autoencoder-Based Gradient Compression

We consider data-parallel distributed training under the parameter server communication protocol, which is one of the most well-established protocols for distributed communication [11]. In this scenario, the nodes are divided into two types: the worker nodes, which contain a replica of the neural network and calculate the gradient tensor of it using the data they have access to, and the master node, which receives the gradient tensors from the worker nodes, performs a reduction operation and sends back the updated gradient tensor to the worker nodes.

A. Common and Innovation Component Model

Under the parameter server distributed training, consider a set of gradient tensors, each unfolded in the form of a vector g_k, $k = 1, \ldots, K$. These gradient vectors correspond to the same training iteration across the K worker nodes, where the index of the iteration is omitted for simplicity. Our empirical results, reported in Section IV, suggest that these gradient vectors are highly correlated, alias, their mutual information (MI) is high. In other words, in distributed training, there is a significant amount of redundant information, which if eliminated can further reduce the communication rates without affecting the performance of the trained model. We model the correlation between the gradient vectors using the following *common and innovation components* model. Specifically, each gradient vector can be expressed as:

$$g_k = g^{cp} + g_k^I, \quad k = 1, 2, \ldots, K, \tag{1}$$

where g^{cp} is the common component that models the information shared by the K gradient vectors and g_k^I is the innovation component that expresses the information unique to each gradient vector. We realise the innovation component as a lightweight signature of the gradient vector; specifically, we set the innovation component of a gradient vector as a vector comprising its top-magnitude values and zeros elsewhere (the sparsity of the resulting vector is 0.001%). Furthermore, we propose an autoencoder (see Section II-B) that learns the common component across the gradient vectors and reconstructs the gradient vectors by combining the learned common component and the innovation component.

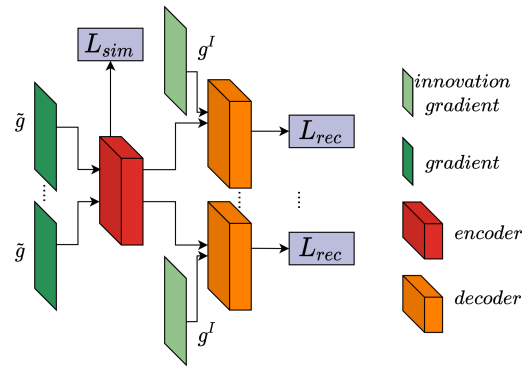

Fig. 1. The architecture of the proposed autoencoder. The gradients are fed sequentially to the encoder and, at each decoder, the innovation gradients are concatenated with the intermediate output before the last convolutional layer.

B. Autoencoder Architecture

The proposed autoencoder for the compression and reconstruction of the gradients consists of one encoder, E_c, and K decoders, D_c^k, $k = 1, \ldots, K$ (see Fig. 1). The encoder encodes one of the gradients g_i, with $i \in (1, \ldots, K)$, into a *compressed common representation*:

$$g^c = E_c(g_i). \tag{2}$$

Each of the K decoders inputs the compressed common representation, g^c, and combines it with the innovation information of the k-th gradient, g_k^I, to obtain the corresponding reconstructed gradient vector,

$$g_k^{rec} = D_c^k(g^c, g_k^I). \tag{3}$$

During the training process of the autoencoder, in order to obtain the compressed common representation g^c from the gradient vectors, all gradients are fed to the encoder E_c. The following similarity loss, i.e., the Euclidean distance between the compressed representations of the gradients, is minimized:

$$L_{sim} = \sum_{k=0}^{K} \sum_{m=0,\, m \neq k}^{K} ||(E_c(g_k) - E_c(g_m))||_2^2. \tag{4}$$

In order to reconstruct the original gradients from the compressed ones, the following reconstruction loss is also applied to the output of the decoders D_c^k:

$$L_{rec} = \sum_{k=0}^{K} ||(g_k - g_k^{rec})||_2^2. \tag{5}$$

The final loss function therefore consists of two terms, the reconstruction loss and the similarity loss, that is,

$$L = \lambda_1 L_{rec} + \lambda_2 L_{sim}. \tag{6}$$

The proposed autoencoder network consists of convolutional and deconvolutional layers. The kernels of the convolutional and deconvolutional layers are one-dimensional (1D) since the inputs of the networks are vectors. This approach reduces the number of network parameters compared with the conventional 2D kernels by approximately 60%. The encoder, E_c, which

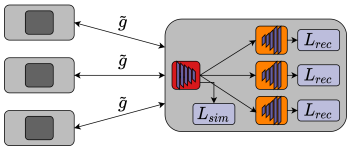

Fig. 2. Training of the proposed autoencoder. $\tilde{g}$ is the gradient vector that is constructed using the top-magnitude gradient values. Note that in the training phase of the autoencoder, the innovation gradient vector $\tilde{g}^I$ is extracted on the master node from the $\tilde{g}$ gradient vector.

TABLE I
PERFORMANCE COMPARISON OF THE LGC FRAMEWORK WITH THE OTHER METHODS OF DISTRIBUTED TRAINING. IN THE CASE OF LGC, THE FIRST NUMBER INDICATES THE VALUE FOR THE NODE THAT SENDS BOTH COMPRESSED COMMON REPRESENTATION AND INNOVATION, AND THE SECOND IS FOR THE REST OF THE NODES.

Training Method	Pixel Accuracy	Gradient Size	Compression Ratio
Baseline	46.3%	120MB	1×
DGC	46.5%	0.29MB	413×
Sparse GD	41%	0.29MB	413×
LGC	46.3%	**0.17/0.16MB**	**693/722**×

takes as input the vector g_k, comprises 5 convolutional layers with 1D kernels and a non-linearity in the form of the leaky-relu. The decoder, D_c, takes as input the compressed representation produced by the encoder and performs an upsampling operation using 5 deconvolutional layers. Before the final convolutional layer, the intermediate representation is concatenated with the innovation component, g_k^I, which consists of the top-magnitude values of g_k and zeros elsewhere.

The training of the autoencoder (see Fig. 2) is performed at the master node as follows. The master node receives the gradient vectors from the worker nodes and passes them sequentially to the encoder E_c. The encoded representations are used to calculate the similarity loss L_{sim}. Furthermore, one of the encoded representations (chosen randomly at each iteration) is combined with the innovation components, g_k^I, $k = 1, \ldots, K$, at the corresponding decoders D_c^k to reconstruct the gradients and compute the L_{rec} loss.

III. THE PROPOSED LGC FRAMEWORK

Assume a system with multiple graphics processing units (GPUs) consisting of K processing nodes. The goal is to train a model with L layers in a data-parallel distributed manner using synchronous stochastic gradient descent (SGD) under the parameter server scenario. Without loss of generality, we assume that the model is a fully convolutional neural network. At each training iteration, a gradient tensor $\nabla_{k,l}$ is produced for each layer $l = 1, \ldots, L$ of the neural network and for each node $k = 1, \ldots, K$. Each such gradient tensor is unfolded in the form of a vector $g_{k,l} \in \mathbb{R}^{n_l}$, where $n_l = k_l^h \cdot k_l^w \cdot f_{l-1} \cdot f_l$, with k_l^h, k_l^w denoting respectively the kernel height and the kernel width at the layer l, f_{l-1} the number of filters in the previous layer and f_l the number of filters in the current layer.

In order to reduce the amount of gradient information sent from each node, a certain amount of values are selected from each vector, $g_{k,l}$. Specifically, the framework extracts the $\alpha\%$ of the values in $g_{k,l}$ with the highest magnitude and constructs the vector $\tilde{g}_{k,l} \in R^{\mu_l}$, where $\mu_l = \frac{\alpha}{100} \cdot n_l$. The top-magnitude gradient selection process is repeated for all layers and concatenates the $\tilde{g}_{k,l}$, $l = 1, \ldots, L$, vectors together to form the vector $\tilde{g}_k \in R^{\mu}$, with $\mu = \sum_{l=0}^{L} \mu_l$. This process is performed independently at each node $k = 1, \ldots, K$ with α fixed across the nodes (typically $\alpha = 0.1$). The transferred indices are entropy encoded—using the DEFLATE compression method [12]—and their rate is taken into account in the total rate calculation. The remaining non-selected gradients are being accumulated at the worker nodes using a momentum correlation similar to the method described in [5].

The encoder E_c at one given worker node k compresses its top-magnitude gradient vector $\tilde{g}_k$ to the representation $\tilde{g}_k^c$, which is in turn transmitted to the master node. In parallel, all worker nodes—including the node mentioned before—apply coarse gradient selection on gradient vectors $\tilde{g}_k$ with an aggressive sparsification rate of 0.001% (we keep 0.001% of the gradients in the tensor), resulting in transmitting the vector $\tilde{g}_k^I \in \mathbb{R}^{0.00001*\mu}$, $k = 1, \ldots, K$ (see Algorithm 1 for more details). One can think of $\tilde{g}_k^I$ as the innovation part of $\tilde{g}_k$, which is specific to each worker node, and $\tilde{g}_k^c$ as the compressed common information shared across all nodes, as per the approach described in Section II.

At the master node, $\tilde{g}^c$ and $\tilde{g}_k^I$ are used to reconstruct the gradient $\tilde{g}_k^{rec}$ with the help of the decoder D_c^k of the proposed autoencoder (see Fig. 3), that is,

$$\tilde{g}_k^{rec} = D_c(\tilde{g}^c, \tilde{g}_k^I). \tag{7}$$

The master node then obtains the aggregated gradient by averaging the reconstructed gradients:

$$\tilde{g}^{rec} = \frac{1}{K} \sum_{k=1}^{K} \tilde{g}_k^{rec}. \tag{8}$$

During the first iterations, the weights of a model (that is trained in a distributed fashion) change very aggressively and thus, the calculated gradients are being rapidly outdated. Any substitution or transformation of the gradients at this stage can be harmful to the performance of the model [5]. For this reason, we do not apply gradient sparsification and compression at the first iterations of the training and instead we use the original gradients. After a number of initial iterations, we update the weights of the model using the gradient vector $\tilde{g}_k$, constructed using the highest magnitude values. In parallel, the autoencoder network is trained at the master node as described in Section II-B. The training of the compression network lasts for a number of iterations and then it can be used to compress the gradients. When the autoencoder is trained, the weights of the learned encoder are transferred to one of the worker nodes. Then, we enter the third stage of training, where the weights are updated with the reconstructed aggregated gradients.

IV. EXPERIMENTS

In this section, we present the evaluation of our LGC framework in the task of compressing the gradients from the

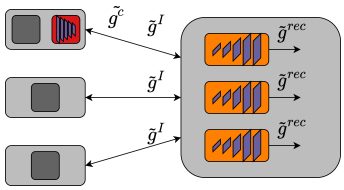

Fig. 3. Distributed training: $\tilde{g}^c$ is the compressed-common gradient vector, $\tilde{g}^I$ is the innovation gradient vector, and $\tilde{g}^{rec}$ is the reconstructed gradient.

Algorithm 1 Compression and reconstruction of the gradient with the LGC on the node k

Input : K nodes, minibatch size b, encoder E_c, decoder D_c^k, Loss function, $Loss$, optimizer SGD, # layers L

$g_{acc} \longleftarrow 0$ **for** *it = 0, 1, ...* **do**
  for l = *0, ...,L* **do**
    $g_l \longleftarrow \nabla Loss + g_{acc}$
    $threshold \longleftarrow min(top\ 0.1\%\ of\ abs(g_l))$
    $mask \longleftarrow abs(g_l) \geq threshold$
    $\tilde{g}_l \longleftarrow mask \odot g_l$
    $g_{acc} \longleftarrow g_{acc} + (\neg mask) \odot g_l$
    $threshold_{inv} \longleftarrow min(top\ 10\%\ of\ abs(\tilde{g}_l))$
    $mask_{inv} \longleftarrow abs(\tilde{g}_l) \geq threshold_{inv}$
    $\tilde{g}_l^I \longleftarrow \tilde{g}_l \odot mask_{inv}$
  end
  $\tilde{g}_k \longleftarrow concatenate(\tilde{g}_l)$
  $\tilde{g}_k^I \longleftarrow concatenate(\tilde{g}_l^I)$
  $\tilde{g}^c \longleftarrow E_c(\tilde{g}_k)$
  $\tilde{g}^{rec} \longleftarrow D_c^k(\tilde{g}^c, \tilde{g}_k^I)$
end

worker to the master node (uplink communication). Similar to prior work [5], [6], downlink communication is outside of the scope of this work, but it can be inexpensive when the broadcast routine is implemented with the "tree" topology as in several Message Passing Interface (MPI) implementations [13]. All experiments are performed on a single machine with four GeForce RTX 2080 Ti GPUs and 128 GB of RAM, by emulating more than one node on each GPU. The LGC framework is built on top of Pytorch's distributed package.

A. Gradient Correlation Analysis

We first analyze the statistical dependencies among the gradient tensors produced by different computing nodes within distributed training using information-theoretic measures; namely, the marginal and conditional entropy, and the mutual information (MI). We conduct experiments using the VGG11 [14] model with the Food101 [15] dataset and the PSPNet [16] model with the CamVid [17] dataset. The trainings are performed on 16 and 2 distributed nodes, respectively. The calculation of the aforementioned measures is based on the histograms of the discretized gradients, which are quantized using the uniform quantizer with 2^{32}-level (32-bit) accuracy. Figure 4 depicts the marginal entropy and the MI for different pairs of layers in the models throughout the training iterations. We observe that the MI values are high; approximately 80%

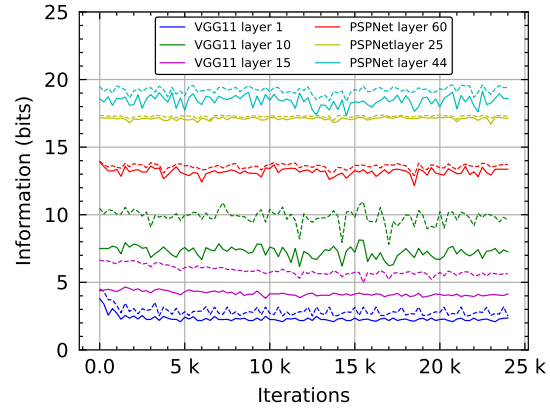

Fig. 4. The mutual information (solid lines) and the marginal entropy (dotted lines) between gradient tensors of the same layer on the different nodes, through the training iterations, for distributed training of the VGG11 and the PSPNet models.

of the average information content (i.e., the entropy) contained in the layer's gradient tensor at every iteration is common for both nodes. These results corroborate our hypothesis that there is a significant amount of information that can be obtained from one gradient tensor about the other per iteration.

B. Distributed Training Results

We now evaluate the performance of the LGC framework in reducing the rate of gradient communication in distributed training. We consider the training of two models: the Resnet50 convolutional neural network trained on the ImageNet [9] dataset (image classification task) and the PSPNet [16] model trained on the CamVid [17] semantic segmentation dataset (image-to-image transformation task).

The distributed training strategy within LGC is as follows: the model is initially trained without gradient modification for 200 iterations. Then, the weights are updated using the top-magnitude values of the gradients, and at the same time, the autoencoder is being trained. This process lasts for another 200 iterations. The autoencoder is trained with the SGD optimizer using a learning rate of 0.001 and a batch size of 1. For the remaining iterations the distributed training is performed with the compressed top-magnitude values of the gradients using the trained autoencoder. For the selection of the top-magnitude gradient values, we set the sparsity parameter to $\alpha = 0.1\%$.

Regarding the hyperparameters used in the trainings, we follow the exact same setup as in the original papers (see [3] for the ResNet model and [16] for the PSPNet model). For the experiments with DGC and Sparse GD, we follow the strategies that were described at the corresponding papers of DGC and Sparse GD. In all experiments, we report the compression ratio (CR) defined as, $\text{CR} = \frac{\text{size}\left(G_k^{\text{original}}\right)}{\text{size}\left(G_k^{\text{compressed}}\right)}$, where G_k^{original} and $G_k^{\text{compressed}}$ are the uncompressed and compressed gradients at the training node k, and the size$(\cdot)$ function computes the size of the gradient tensor in Megabytes.

TABLE II
PERFORMANCE IN DISTRIBUTED TRAINING OF RESNET50 ON IMAGENET. IN THE CASE OF LGC, THE FIRST NUMBER IN THE COMPRESSION RATIO INDICATES THE RATE FOR THE NODE THAT SENDS BOTH THE COMPRESSED COMMON AND THE INNOVATION COMPONENT, AND THE SECOND NUMBER IS FOR THE REST OF THE NODES.

Training Method	Top1 Accuracy	Compression Ratio	Total Information
Baseline	75.98%	1×	351TB
LGC	75.88%	**386/2800×**	**0.4TB**
ScaleCom	75.98%	96×	3.6TB
DGC	76.01%	277×	1.2TB
Sparse GD	75.54%	277×	1.2TB

We conduct distributed training of ResNet50 on ImageNet [9] on 8 nodes and assess the achieved accuracy and speedup versus the gradient compression ratio. We used the following settings: the SGD optimizer with a momentum of 0.9, a weight decay of $1e-4$, and an initial learning rate of 0.1 that decays by 10 every 30 epochs. The reported results are single-crop performance on the ImageNet validation set. Table II depicts the Top1 accuracy versus compression ratio as well as the total amount of the gradient information (in TBs) sent from all nodes during the whole training for the LGC framework and alternative state-of-the-art methods, namely, DGC [5] (our implementation), ScaleCom [10] and Sparse GD [4] (our implementation). The results show that the proposed framework is able to achieve substantial compression ratios—namely, $386\times$ compression of the gradients for the node transmitting the common component and its innovation component and $2800\times$ for the nodes transmitting only their innovation component—without loss of accuracy compared with the baseline method, which performs distributed training with the uncompressed, non-modified gradients. In Table II, the reported total amount of gradient information transferred during the entire training in the case of the LGC, includes updates with the original and the top-k gradients during the first two training phases, as described in Section III. The total amount of information sent is lower by $794\times$ compared with the baseline training. Furthermore, we achieved $1.7\times$ speedup over the baseline. The duration of one iteration for each type of gradient update that we are using within a distributed training is the following: *(i)* 1 sec for the updates with the uncompressed gradients; *(ii)* 1.6 sec for the updates with the top-k gradients; and *(iii)* 0.6 sec for the updates with the compressed gradients.

We also perform distributed training of the PSPNet model on 2 nodes and assess the achieved pixel accuracy versus the gradient compression ratio. The model is trained on the CamVid [17] dataset, which consists of 32 different classes and contain 701 images with a spatial resolution of 720×960 pixels. In this experiment, we use a batch size of 12 and momentum SGD. The results, which are reported in Table I, show that our proposed LGC framework can reduce the size of the gradient information sent from each node at each iteration by $693\times$ for the node transmitting the common component and its innovation component and $722\times$ for the nodes transmitting only their innovation component, without a reduction in the pixel accuracy. Furthermore, the proposed LGC framework achieves a significantly improved compression ratio compared to alternative state-of-the-art methods for distributed training such as DGC [5] and Sparse GD [4].

V. CONCLUSION

In this paper, we have introduced a novel method for data-parallel distributed training of deep neural networks. We have shown empirically that our LGC framework can provide up to $794\times$ reduction in the total number of bits being transferred within a distributed training of ResNet50 on ImageNet, compared with the baseline distributed training with original uncompressed gradients. This was made possible by exploring the correlations between the gradients of different nodes within the scope of distributed training and designing distributed autoencoder for gradient compression.

REFERENCES

[1] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker *et al.*, "Large scale distributed deep networks," in *NIPS*, 2012.

[2] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training," *Proceedings of the VLDB Endowment*, vol. 13, no. 12.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[5] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proceedings of the ICLR*, 2018.

[6] D. Basu, D. Data, C. Karakus, and S. Diggavi, "Qsparse-local-SGD: Distributed sgd with quantization, sparsification, and local computations," 2019. [Online]. Available: https://arxiv.org/pdf/1906.02367.pdf

[7] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *NIPS*, 2017, pp. s 1707–1718.

[8] A. Abdi and F. Fekri, "Reducing communication overhead via CEO in distributed training," in *IEEE International Workshop on SPAWC*, 2019.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei., "ImageNet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[10] C.-Y. Chen, J. Ni, S. Lu, X. Cui, P.-Y. Chen *et al.*, "Scalecom: Scalable sparsified gradient compression for communication-efficient distributed training," *Advances in Neural Information Processing Systems*, 2020.

[11] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[12] P. W. Katz, "String searcher, and compressor using same," *US Patent*, no. 5051745, 1991.

[13] M. J. Rashti, J. Green, P. Balaji, A. Afsahi, and W. Gropp, "Multi-core and network aware mpi topology functions," in *European MPI Users' Group Meeting*. Springer, 2011, pp. 50–60.

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[15] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101 – mining discriminative components with random forests," in *European Conference on Computer Vision*, 2014.

[16] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[17] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A highdefinition ground truth database," *Pattern Recognition Letters*, 2009.