IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 7 July 2022, accepted 29 July 2022, date of publication 8 August 2022, date of current version 16 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3197646

RESEARCH ARTICLE

AFSD: Adaptive Feature Space Distillation for Distributed Deep Learning

SALMAN KHALEGHIAN[1], HABIB ULLAH[2], EINAR BROCH JOHNSEN[3], ANDERS ANDERSEN[1], AND ANDREA MARINONI[1], (Senior Member, IEEE)

[1]Faculty of Science and Technology, UiT The Arctic University of Norway, 9019 Tromsø, Norway
[2]Faculty of Science and Technology, Norwegian University of Life Sciences (NMBU), 1430 Ås, Norway
[3]Department of Informatics, University of Oslo, 0315 Oslo, Norway

Corresponding author: Salman Khaleghian (salman.khaleghian@uit.no)

This work was supported in part by the Centre for Integrated Remote Sensing and Forecasting for Arctic Operations (CIRFA) and the Research Council of Norway (RCN) under Grant 237906, in part by the European Union's Horizon 2020 Research and Innovation Programme Extreme Earth Project under Agreement 825258, and in part by the Fram Center under the Automised Large-Scale Sea Ice Mapping (ALSIM) through the "Polhavet" Flagship Project.

ABSTRACT We propose a novel and adaptive feature space distillation method (AFSD) to reduce the communication overhead among distributed computers. The proposed method improves the Codistillation process by supporting longer update interval rates. AFSD performs knowledge distillates across the models infrequently and provides flexibility to the models in terms of exploring diverse variations in the training process. We perform knowledge distillation in terms of sharing the feature space instead of output only. Therefore, we also propose a new loss function for the Codistillation technique in AFSD. Using the feature space leads to more efficient knowledge transfer between models with a longer update interval rates. In our method, the models can achieve the same accuracy as Allreduce and Codistillation with fewer epochs.

INDEX TERMS Distributed deep learning, convolutional neural networks, knowledge distillation, codistillation.

I. INTRODUCTION

To efficiently process big data, new deep learning based systems have been proposed. These systems significantly improve the overall performance when considering the big data. Furthermore, these systems scale up the training and inference process of deep learning techniques. To address the need for computational resources, the training process could be distributed across multiple computers connected by a network [1], [2]. Distributed deep learning is the most widely used approach to speed up the neural network training by leveraging the computational resources of multiple devices (e.g., multiple GPUs) [1]. These devices are used to accelerate training by distributing data (data-parallel) across the devices. Each device holds a copy of the model being trained, and the copies are kept synchronized throughout the training process. In one step of a typical implementation, every device computes a gradient using different data samples. The gradients are then averaged across all devices (e.g., via an Allreduce operation). Subsequently, each device locally performs an optimization step using the average gradient [1], [3], [4]. The model parameters on every device are initialized to the same value to keep them synchronized. Increasing the number of devices brings more computational power. However, it also brings a significant communication overhead to synchronize the model parameters across all devices at every step [5], [6].

In the literature, different methods are introduced to reduce the communication overhead. For example, quantizing or compressing gradients before synchronizing them [7], synchronizing periodically rather than on every update [8], and only synchronizing among subsets of devices [9]. These methods reduce the communication overhead per update but they also impact the quality of a model or training time. In this regards, one-step Codistillation [10] method is proposed based on online distillation. The distillation process involves training a single model to match the ensemble output rather than the real data labels [11]. Codistillation makes use of distillation in an online manner to accelerate training by

The associate editor coordinating the review of this manuscript and approving it for publication was Massimo Cafaro.

transferring the improved performance of the ensemble to each model [12]. In the codistillation technique, the update interval for synchronizing the stored models is important, because it can affect the communication overhead between the machines. However, by having a longer update interval, the efficiency of knowledge distillation is crucial for knowledge sharing between models. Otherwise, the training time would increase which is against the essence of the Codistillation process.

In this paper, we propose *adaptive feature space distillation* (AFSD). In our proposed method, the models perform knowledge distillation by sharing features instead of output as in regular Codistillation [10]. We achieve this by means of a new loss function for the Codistillation technique, characterized by reduced communication overhead. Our method can achieve the same accuracy with longer update intervals by considering fewer epochs. Our method performs knowledge distillation across the models infrequently, which provides flexibility to the models in terms of learning diverse variations in the data. In short, the main contribution of this paper are:

- A novel and adaptive feature space distillation method characterized by reduced communication overhead.
- A new loss function for knowledge distillation which shares features instead of output in the Codistillation technique.
- We outperform the state-of-the-art methods, Allreduce and Codistillation, by getting the same performance with fewer epochs.

The paper is organized as follows: related work is discussed in Section II and background material in Section III. We present the AFSD method in Section IV. Section V validates the proposed method experimentally. Section VI discuss the conclusions and future work.

II. RELATED WORK

We can categorize distributed deep learning techniques from two perspectives [1], namely *concurrency in networks* and *concurrency in training*. The first category can be further divided into the two sub-categories: *model parallelism* and *data parallelism*.

A. CONCURRENCY IN NETWORKS

In this category, we compute the output of the layers or the whole network in concurrent mode for the forward evaluation and backpropagation phases. Model parallelism divides the work according to the neurons in each layer. Different parts of the Deep Neural Network (DNN) are computed on different processors in different machines [1]. For example, Huang *et al.* [13] proposed an approach for training huge DNNs that can not be stored in one GPU. With data parallelism, several replicas of a neural network model are created during training, each on a different worker (processor). The workers process different mini-batches locally at each step using an optimizer. For example, the replicas of the model are synchronized (i.e., either by average gradients or parameters) at every step by communicating either with a centralized parameter server [14], [15] or decentralized using Allreduce [16], [17], [18]. By relaxing the synchronization restrictions and creating an inconsistent model, training workers can read parameters and update gradients asynchronously [19]. Data communication in distributed deep learning can be reduced using methods such as quantization [20], [21], [22] or sparsification [23], [24], [25], [26], [27].

B. CONCURRENCY IN TRAINING

In this category, concurrency is used in the training stage. Multiple instances of training processes run independently on different machines. Concurrency is also used for ensemble learning. Distributed training of ensembles is a completely parallel process, requiring no communication between the workers [28]. Ensemble learning requires more memory and computational power in the training and inference phases. Therefore, knowledge distillation has been used in a two-step training to transfer knowledge of an ensemble with several networks to a single network [12], [29], [30]. To handle the problem of two-step training, Zhang *et al.* [31] investigated how an ensemble of students can learn collaboratively and teach each other throughout the training process. Kim *et al.* [32] introduced a fusion learning method that trains a robust classifier by integrating feature maps. Park and Kwak [33] used feature-level ensembles for knowledge distillation by transferring the ensemble knowledge between multiple teacher networks. Although these methods can be trained in parallel, their main problem is accuracy when the number of epochs is not taken into account. Codistillation [10] taken advantage of ensemble learning and mutual learning to speed up the training. Codistillation uses a distillation-like loss that penalizes predictions made by one model on a batch of training samples for deviating from the predictions made by other models on the same batch.

Our proposed method falls in the category *concurrency in training*. It includes distilled knowledge between models by directly tuning their feature space.

III. BACKGROUND

Distributed ensemble learning (DEL) addresses the problem of communication overhead by training multiple instances of models (weights) independently on the same dataset. The overall prediction is the average of the predictions of all the models. DEL requires no communication between the computers [1]. However, ensemble learning increases the cost during the validation stage since the predictions from multiple machines are averaged. Ensemble learning also causes latency [1]. The distillation approach [12] addresses this problem by a two-step processes. In the first step, ensemble learning is performed over several machines, so-called teachers. In the second step, a student model is trained to mimic the teacher models. The student model is then used during the test stage. It reduces the cost of ensemble learning by adding another phase to the training process. Using more machines, distillation increases the training time and

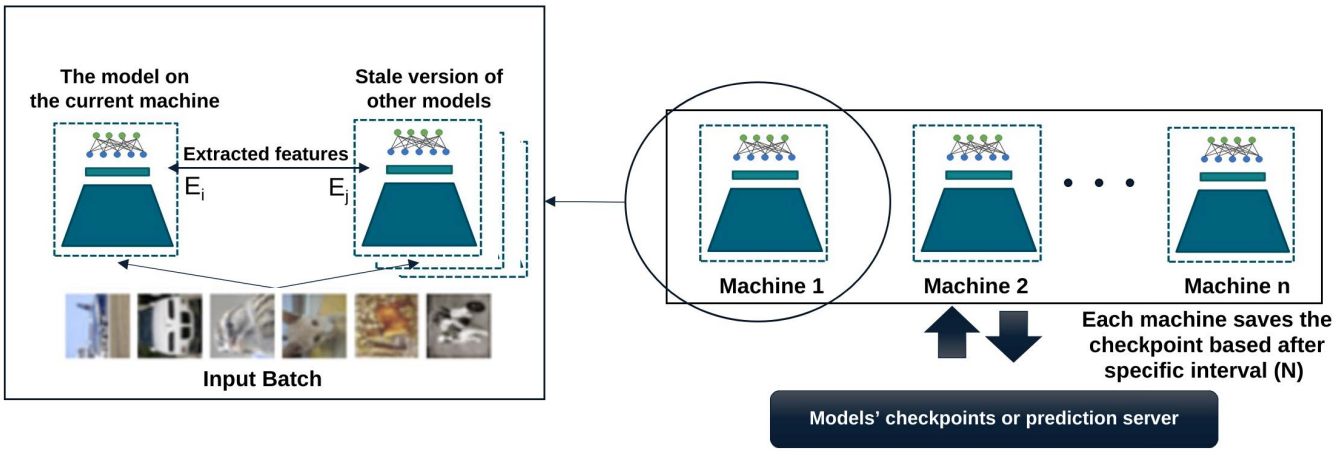

FIGURE 1. Architecture of our proposed method. The checkpoints of the models are stored on a shared storage or a prediction server after the specific interval (N). Each model is forced to produce the same feature space as the others models for the same inputs batch.

complexity in return for a quality improvement close to a larger teacher ensemble model [10].

However, the ensemble method with distillation remains time consuming. In contrast, one-step Codistillation [10] is based on online distillation and trains N copies of a model in parallel. It starts distillation early in the training process. In the Codistillation technique, the length of the update interval can affect the communication overhead among the machines. For example, the longer the update interval, the lower is the communication overhead. In the ideal case, the communication among different models should be reduced with longer update intervals. Moreover, the update interval affects the diversity between the trained models. The distillation process in Codistillation reduces diversity by forcing the models to predict the same outputs for the same inputs.

IV. AFSD: ADAPTIVE FEATURE SPACE DISTILLATION

AFSD, the method proposed in this paper, exploits the feature space of each model to explore more variations in the training data instead of using the outputs of the networks to share knowledge between the models. In fact, we tune the models to generate similar feature spaces for transferring knowledge between the models. We consider feature spaces to be similar when the distance between extracted features is the same for the same inputs using different models. To perform knowledge distillation more efficiently with fewer epochs, we manipulate and tune the feature space by considering a distillation term. Our method is based on the Codistillation technique [10] to share knowledge between models rather than synchronizing models to have the same weights. We train n copies of a model in parallel and start distillation early in the training process by adding a new distillation loss term to the loss function. In fact, we have a set of students who simultaneously learn during the training process to handle the classification together (Figure 1). Each model saves the checkpoint of its weights on a shared storage after each update interval, so each model considers the other models as teachers in a distillation-like setup. However, each model uses the stale version of stored models on shared storage (or a prediction server) and performs additional forward passes with the same input batch.

A. FORMULATION OF THE LEARNING PROCESS

We consider an input batch $X = \{x_1, \ldots x_n\}$ where x_k is an input sample and n is the size of the input batch and labels $Y = \{y_1, y_2, \ldots, y_n\}$ according to the input batch X. The output of model i is defined by $f_{\theta i}(x_k)$ where $x_k \in X$ and θ_i is the parameter of model i. The extracted features from model i are represented by $F_{\theta i}(x_k) = a_{ik}$ where a_{ik} are extracted features by considering the x_k sample by the model i. The number of models is represented by m.

We propose a loss function including a distillation penalty to force the models to produce the same feature space (extracted features before fully connected layers (FC)). For this purpose, the penalty term forces models to have the same distance between extracted features when considering the same inputs. In other words, if the distance between two features for two samples is shorter or longer using one model, the distance between the features for the same two samples extracted from the other models should be similar. For this purpose, we formulate a $n \times n$ distance matrix representing the distance between an individual feature and all other features. We formulate the distance matrix E_i of the model i as

$$E_i = \begin{pmatrix} D(a_{i1}, a_{i1}) & D(a_{i1}, a_{i2}) & \ldots & D(a_{i1}, a_{in}) \\ D(a_{i2}, a_{i1}) & D(a_{i2}, a_{i2}) & \ldots & D(a_{i2}, a_{in}) \\ \ldots & \ldots & \ldots & \ldots \\ D(a_{in}, a_{i1}) & D(a_{in}, a_{i2}) & \ldots & D(a_{in}, a_{in}) \end{pmatrix} \quad (1)$$

where a_{ik} and a_{ij} are extracted features considering samples x_k and x_j using model i. Let D be the distance metric and n the input batch size. We formulate the loss function for the model i as

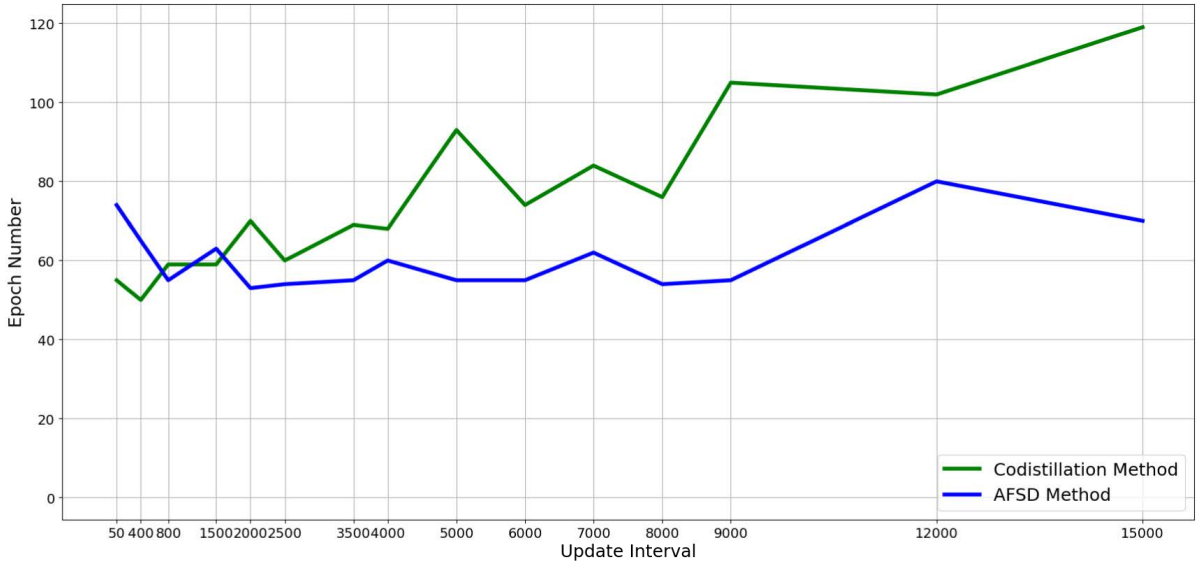

FIGURE 2. Comparison between earliest epoch number that achieves the same accuracy as Allreduce considering Codistillation and our proposed method.

follows:

$$loss = L(Y, f_{\theta i}(X)) + \alpha \frac{1}{m-1} \sum g(E_i, E_j) \quad i \neq j \quad and \quad j \in 1, 2, 3, \ldots m \tag{2}$$

$$g(E_i, E_j) = \frac{1}{n} e^T |E_i - E_j| e \tag{3}$$

where m is the number of models on different machines, X is the input batch, Y is the input labels, α is the penalty coefficient and L represents the loss between prediction and the labels. The function g indicates the average distance between elements of E_i and E_j considering batch size n, where e is the column vector whose entries are all 1's and T is the transpose operator. The first term is the cross entropy loss and the second term is the distillation loss. Based on this loss function, we show our proposed method in Algorithm 1.

Algorithm 1 AFSD Algorithm

1: **Initialization** of network parameters $(\theta_i, i \in 1, 2, 3, \ldots, m)$
2: **Initialization** of learning rate μ and penalty coefficients α for each model
3: **Repeat** for the number of epochs:
4: **Do in parallel** for $i \in 1, 2, 3, \ldots, m$:
5: **Get next batch** (X,Y) by size n
6: **Update** θ:
$\theta_i^{k+1} = \theta_i^k + \mu \nabla_{\theta i}(L(Y, f_{\theta i}^k(X)) + \alpha \frac{1}{m-1} \sum g(E_i, E_j))$
i ≠ j and j ∈ 1,2,3, ...m

V. EXPERIMENTAL ANALYSIS

In our experiments, we want to evaluate our method and show that it can address the aforementioned problems. The evaluation is done in terms of four research questions (RQs). RQ1: How does AFSD cope with the impact of the update interval without affecting the performance? RQ2: Does AFSD achieve the same performance with fewer epochs considering a longer update interval? RQ3: How does the new distillation loss term based on the feature space affect the training process and the outputs of the models? RQ4: How does AFSD work when different network architectures are trained? We address these research questions in the subsections V-A, V-B, V-C, respectively.

Experimental Setup and Design: In our experiments, we use the standard CIFAR10 dataset [34]. For comparison, we consider the Allreduce [16] and the Codistillation [10] techniques. We consider the Allreduce technique as a baseline for comparison. For the Allreduce technique, we used hyperparameters and the gradual warmup strategy for changing the learning rate [16]. For evaluating our model, we also consider different architectures, namely ResNet20, ResNet32, and VGG16. In case of ResNet20, we trained the network using the Allreduce technique and we achieved a top-1 validation accuracy of 91.71% after 114 epochs. In the Allreduce technique, the input batch is divided

TABLE 1. Validation accuracy and earliest epoch that achieves the same accuracy as Allreduce considering Codistillation [10] and AFSD with update interval between 40 and 3500.

	50		400		800		1500		2000		2500		3500	
	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy
Codistillation [10]	55	91.78	50	91.71	59	91.71	59	91.79	70	91.71	60	91.81	69	91.73
AFSD	74	91.77	65	91.74	55	91.74	63	91.72	53	91.74	54	91.75	55	91.74

TABLE 2. Validation accuracy and earliest epoch that achieves the same accuracy as Allreduce considering Codistillation [10] and AFSD with update interval between 4000 and 15000.

	4000		5000		6000		7000		8000		9000		12000		15000	
	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy
Codistillation [10]	68	91.72	93	91.76	74	91.72	84	91.72	76	91.72	105	91.72	102	91.71	119	91.76
AFSD	60	91.74	55	91.96	55	91.72	62	91.71	54	91.79	55	91.71	80	91.76	70	91.73

between different GPUs. However, we use the same batch input for training the models based on AFSD. Therefore, we feed the data twice as compared to the Allreduce technique and we expect that AFSD driven by linear scalability should achieve the same accuracy with half the number of epochs. We set the batch size such that each GPU receives 128 samples in each batch in all three methods.

In our experiments, we use two servers with three Nvidia GPUs namely Quadro RTX 5000 with 16GB memory on each. The servers are connected through a point-to-point 10Gb network. We use NFS shared storage to save and restore models checkpoints.

A. UPDATE INTERVAL (RQ1 AND RQ2)

We compare our proposed method with the Codistillation method [10] considering different update intervals. We consider update intervals from 50 steps to 15000 steps. To show the capability of our proposed method, we compare the epoch number on which each method achieves the desired accuracy based on the Allreduce method. The comparison between the Codistillation method and our proposed method is shown in Figure 2. Table 1 shows validation accuracy and earliest epoch that achieves the same accuracy as Allreduce considering Codistillation [10] and AFSD with update interval 40 to 3500 and Table 2 shows same validation accuracy for update interval from 4000 to 15000. In this experiment, we record the epoch number when a specific method reaches the same accuracy as Allreduce. It can be seen, when the update interval is longer, our method achieves the desired accuracy with much fewer epochs. Additionally, our proposed method is driven by linear scalability and tolerates longer update intervals considering 12000 steps. In fact, when using 9000 steps as the update interval, we update the saved checkpoint after 23 epochs (the batch size is 128, and we have 50000 samples in the training dataset). This shows that our method can achieve the same accuracy and scalability with very little communication overhead. However, when we update the models more often with shorter update intervals, we reduce the diversity of the models. When we use more powerful distillation, information sharing does not have the intended benefits. The comparison between Allreduce, Codistillation, and our proposed method using ResNet20 [36] is shown in Figure 3. We consider update intervals equal to 5000 and 9000 steps. We use the early stopping strategy when Codistillation and AFSD achieve 91.71% or more accuracy. We reduce the learning rate on epochs 45 and 55 with a factor 0.1 when Codistillation and AFSD are used. We also consider the first four epochs as the warm-up epochs applicable to the update interval. Otherwise, we let the networks continue the training independently, based on the specified update intervals. As we can see in Figure3, our method can achieve the same accuracy with fewer epochs compared to Allreduce and Codistillation. Since we are using much longer update intervals, the networks are trained more independently in a different direction. Therefore, we can see a rise in training loss when we transfer knowledge based on distillation loss terms. However, the loss reduces through the training process.

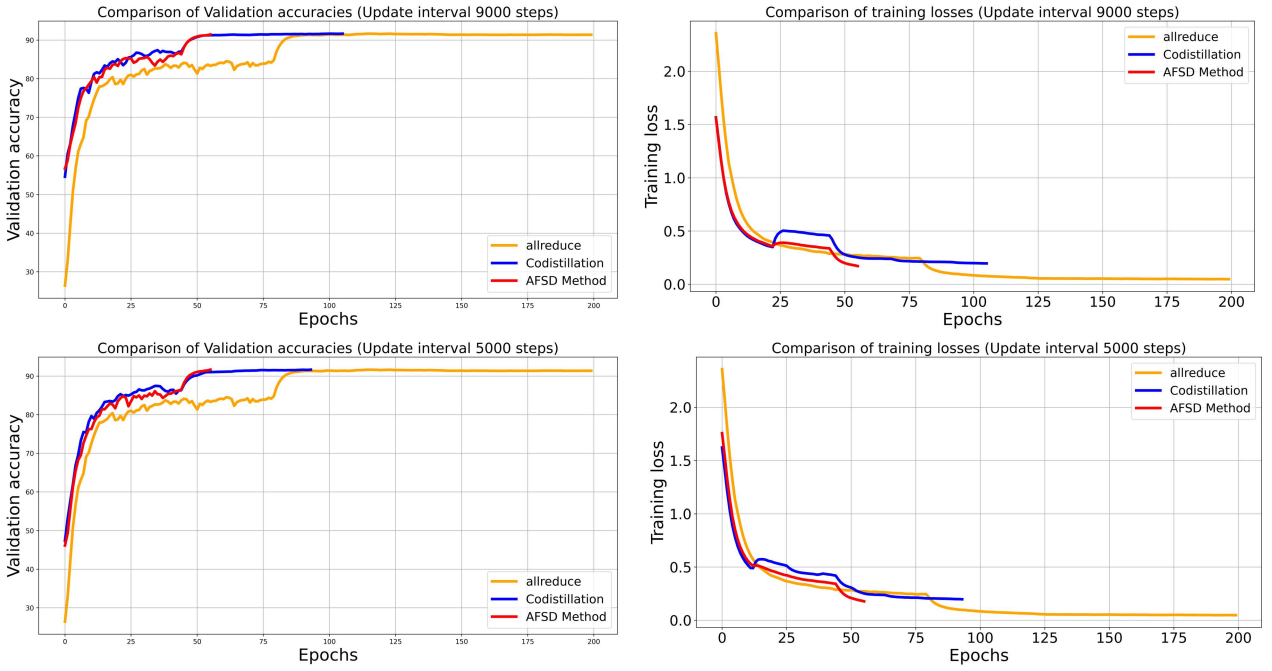

FIGURE 3. Validation accuracy and training loss using Codistillation [10], our proposed method, and Allreduce [35]. We use the early stopping strategy when the Codistillation achieves the same accuracy of Allreduce considering 9000 steps (a, b) and 5000 steps (c, d) update intervals.

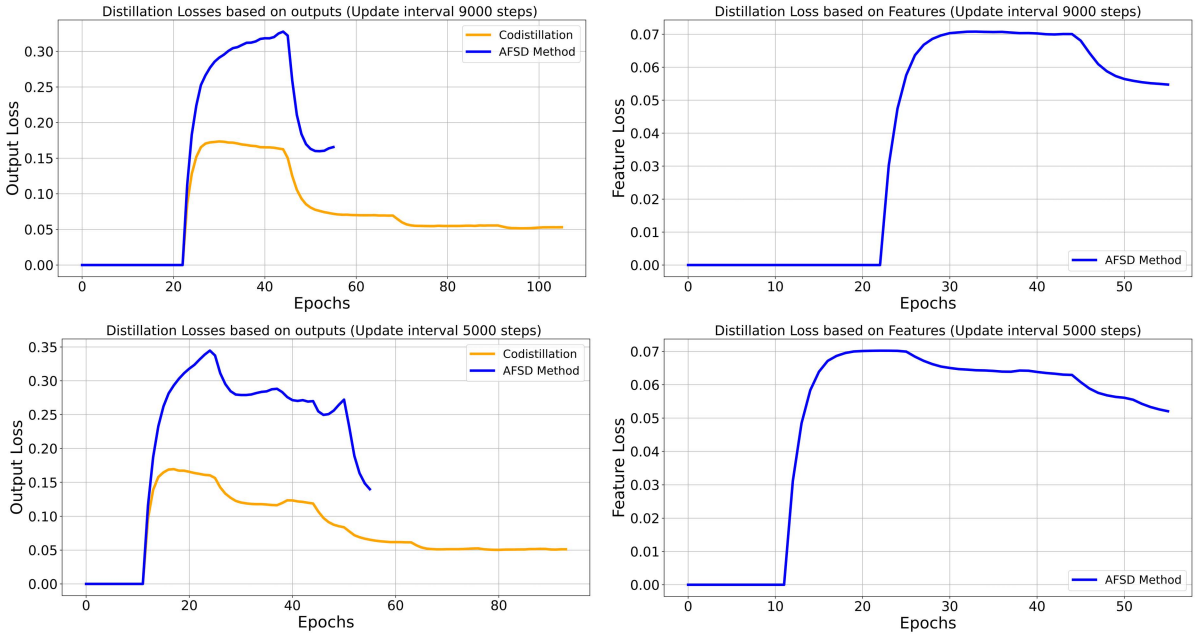

FIGURE 4. Distillation losses based on the outputs and features considering Codistillation and our proposed method. We do not include distillation loss based on the outputs in our method, but we measure it to compare with the Codistillation approach.

B. DISTILLATION LOSS (RQ3)

The loss function is based on two terms: cross-entropy loss and distillation loss. The Codistillation technique uses distillation loss based on the output of the networks. In contrast, our method encodes distillation loss based on the feature space. We want to illustrate the difference between the two loss terms by showing how these behave during the training process for AFSD. Figure4 shows the distillation loss considering the Codistillation method and AFSD. It should be noted, in our method, we do not use output-based distillation, but we measure it during the training. As we can see, in Figures4 (a) and4 (c), distillation based on outputs fluctuates

and even increases with AFSD. The networks are robust for classifying the extracted features even with different outputs since we just force them to generate the same features. This is because the neural network models can represent the same function in different ways with different parameter values [37]. However, Figures4 (b) and4 (d) show that the loss between extracted features is reduced through the training, and we can transfer knowledge between the models. Even small changes in the features can lead to more effective distillation, and the networks can achieve the same accuracy with fewer epochs.

C. NETWORK ARCHITECTURES (RQ4)

In order to evaluate the generalization capability of AFSD regardless of the use of a specific network architecture, we consider other architectures in this section. Figure5 shows the validation accuracy of the ResNet32 network using Codistillation and AFSD for update intervals equal to 5000 and 9000 steps. For this experiment, we consider 92.41% as the top-1 accuracy of Allreduce. The Allreduce operation achieves this accuracy after 123 epochs. We use the learning rate schedule for this network to reduce the learning rate by a factor of 0.1 on the epoch number equal to 50 and 60. As we can see, our method achieves this accuracy with fewer epochs compared to the Codistillation method.

We also explore the VGG-16 [38] model and a 13-layer CNN [39] architecture to consider architectures not belonging to the ResNet families. However, considering both Codistillation and AFSD, using these architectures, we would not get the same accuracy with fewer epochs then needed for the Allreduce technique. Therefore it seems these methods can be more effective with the ResNet family of architectures.

D. THREATS TO VALIDITY

In our experiments, we used three GPUs on each server. In each server we considered the Allreduce algorithm to train a model in a synchronized manner on these three GPUs. Increasing the number of GPUs on each server could affect the results since it would increase the number of epochs to get the appropriate accuracy. Since this would be same for both Codistillation and AFSD, we consider them significant parameters. Experiments with more GPUs on each server in a two-way setup can be considered.

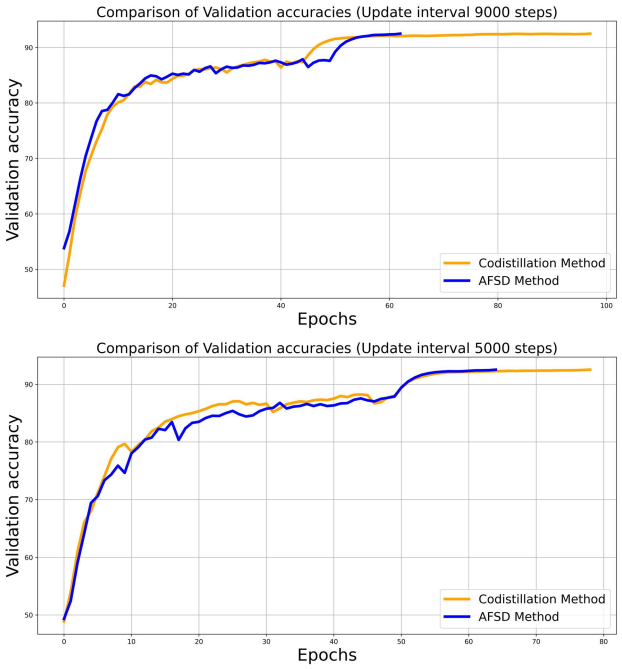

FIGURE 5. Validation accuracy of the ResNet32 network using Codistillation and our proposed method for the update intervals equal to 5000 and 9000.

We considered the ResNet20, ResNet30, VGG16 and a 13-layer CNN [39] architectures in our experiments. Deeper architectures with more parameters could exhibit different behaviors. Deeper architectures usually learn features at various levels of abstractions. Therefore, considering only high-level features at the end of the network would not be sufficient to share knowledge. We can also observe this issue when a pure convolution network like VGG16 is used. Hence deeper architectures with more parameters and more intermediate features can be considered for future experiments.

We consider random initialization as a diversity enforcement regularization. However, it can violate a specific situation when the initialized weights or training directions are aligned together. Hence we assume that this diversity can be accomplished by weight randomization.

VI. CONCLUSION

In this paper, we propose AFSD, a new method for large scale distributed deep learning. The main

novelty of AFSD is knowledge sharing based on the feature space of parallel models in the Codistillation setup. Our method supports much longer update intervals using a new knowledge distillation loss function. By prolonging the update interval, the models become more diverse and contribute more to the training process. Additionally, the communication overhead is significantly reduced. We show that with only two updates through the training process, the models can achieve linear scalability using the feature space for sharing the information.

In future work, we will consider our approach of feature space sharing for scalable semi-supervised learning. It has been shown that generating pseudo-labels for unlabeled data using feature spaces increases the performance of deep semi-supervised learning [40], [41]. An extension of our method to semi-supervised learning, could be useful to address the issue of scarce training data, which is critical in many areas where a small number of labeled data and a large amount of unlabeled data are available.

REFERENCES

[1] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 1–43, Jul. 2020.

[2] A. C. Zhou, B. Shen, Y. Xiao, S. Ibrahim, and B. He, "Cost-aware partitioning for efficient large graph processing in geo-distributed datacenters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1707–1723, Jul. 2020.

[3] R. Mayer and H.-A. Jacobsen, "Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools," *ACM Comput. Surv.*, vol. 53, no. 1, pp. 1–37, Jan. 2021.

[4] M. A. Soyturk, P. Akhtar, E. Tezcan, and D. Unat, "Monitoring collective communication among GPUs," 2021, *arXiv:2110.10401*.

[5] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," 2016, *arXiv:1604.00981*.

[6] F. Zhang, Z. Chen, C. Zhang, A. C. Zhou, J. Zhai, and X. Du, "An efficient parallel secure machine learning framework on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 9, pp. 2262–2276, Sep. 2021.

[7] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Advances in Neural Information Processing Systems*, vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/6c340f25839e6acdc73414517203f5f0-Paper.pdf

[8] S. U. Stich, "Local SGD converges fast and communicates little," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–19. [Online]. Available: https://openreview.net/forum?id=S1g2JnRcFX

[9] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 344–353.

[10] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton, "Large scale distributed neural network training through online distillation," 2018, *arXiv:1804.03235*.

[11] Z. Allen-Zhu and Y. Li, "Towards understanding ensemble, knowledge distillation and self-distillation in deep learning," 2020, *arXiv:2012.09816*.

[12] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS), Deep Learn. Workshop*, 2014. [Online]. Available: https://openreview.net/forum?id=S1g2JnRcFX

[13] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 103–112.

[14] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proc. NIPS*, vol. 2, 2014, pp. 1–4.

[15] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing, "GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server," in *Proc. 11th Eur. Conf. Comput. Syst.*, Apr. 2016, pp. 1–16.

[16] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," *CoRR*, vol. abs/1706.02677, pp. 1–12, Jun. 2017.

[17] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," 2017, *arXiv:1705.09056*.

[18] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, "S-Caffe: Co-designing MPI runtimes and caffe for scalable deep learning on modern GPU clusters," in *Proc. 22nd ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2017, pp. 193–205.

[19] F. Niu, B. Recht, C. Ré, and S. J. Wright, "HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent," 2011, *arXiv:1106.5730*.

[20] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

[21] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, Y. Bengio and Y. LeCun, Eds., San Juan, PR, USA, May 2016, pp. 1–14.

[22] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

[23] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Proc. Conf. Empirical Methods Natural Lang. Process.* Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 440–445. [Online]. Available: https://aclanthology.org/D17-1045

[24] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "Adacomp: Adaptive residual gradient compression for data-parallel distributed training," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 1–9.

[25] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, Vancouver, BC, Canada, Apr./May 2018, pp. 1–14. [Online]. Available: https://openreview.net/forum?id=SkhQHMW0W

[26] C. Renggli, S. Ashkboos, M. Aghagolzadeh, D. Alistarh, and T. Hoefler, "SparCML: High-performance sparse communication for machine learning," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2019, pp. 1–15.

[27] P. D'Ambra and S. Filippone, "A parallel generalized relaxation method for high-performance image segmentation on GPUs," *J. Comput. Appl. Math.*, vol. 293, pp. 35–44, Feb. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037704271500254X

[28] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra, "Why m heads are better than one: Training a diverse ensemble of deep networks," 2015, *arXiv:1511.06314*.

[29] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan, "Augmix: A simple data processing method to improve robustness and uncertainty," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*, Addis Ababa, Ethiopia, Apr. 2020, pp. 1–15. [Online]. Available: https://openreview.net/forum?id=S1gmrxHFvB

[30] T. Fukuda, M. Suzuki, G. Kurata, S. Thomas, J. Cui, and B. Ramabhadran, "Efficient knowledge distillation from an ensemble of teachers," in *Proc. Interspeech*, Aug. 2017, pp. 3697–3701.

[31] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4320–4328.
[32] J. Kim, M. Hyun, I. Chung, and N. Kwak, "Feature fusion for online mutual knowledge distillation," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, Jan. 2021, pp. 4619–4625.
[33] S. Park and N. Kwak, "Feature-level ensemble knowledge distillation for aggregating knowledge from multiple networks," in *Proc. ECAI*. Amsterdam, The Netherlands: IOS Press, 2020, pp. 1411–1418.
[34] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, USA, Tech. Rep., 2009.
[35] 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," 2017, *arXiv:1706.02677*.
[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[37] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, Y. Bengio and Y. LeCun, Eds., San Diego, CA, USA, May 2015, pp. 1–14.
[39] A. Tarvainen and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1195–1204.
[40] A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, "Label propagation for deep semi-supervised learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 5070–5079.
[41] S. Khaleghian, H. Ullah, T. Kraemer, T. Eltoft, and A. Marinoni, "Deep semisupervised teacher–student model based on label propagation for sea ice classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 14, pp. 10761–10772, 2021.

SALMAN KHALEGHIAN received the bachelor's degree in applied mathematics/computer science and the M.S. degree in computer software engineering, in 2006 and 2010, respectively. He is currently pursuing the Ph.D. degree in scalable computing for earth observation with the Center for Integrated Remote Sensing and Forecasting for Arctic Operations (CIRFA), Faculty of Science and Technology, University of Tromsø (UiT), and the SIRIUS Laboratory, Department of informatics, University of Oslo (UiO). His research interests include machine learning, deep learning, scalable deep learning, and computer vision.

HABIB ULLAH received the M.S. degree in electronics and computer engineering from Hanyang University, South Korea, in 2009, and the Ph.D. degree in information and communication technology from the University of Trento, Italy, in 2015. He served as an Assistant Professor in electrical engineering at COMSATS University Islamabad, Pakistan, from 2015 to 2016. He worked as an Assistant Professor at the College of Computer Science and Engineering, University of Ha'il, Saudi Arabia, from 2016 to 2020. He also worked as a Postdoctoral Researcher at the UiT The Arctic University of Norway, in 2020. He is currently working as an Associate Professor with the Norwegian University of Life Sciences (NMBU). His research interests include computer vision and machine learning.

EINAR BROCH JOHNSEN is currently a Professor with the Department of Informatics, University of Oslo. He is active in formal methods for distributed and concurrent systems, including object-oriented and actor languages, manycore computing, cloud computing, and digital twins. He is one of the main developers of the ABS Modeling Language.

He is the Strategy Director of SIRIUS, a Center for Research-Driven Innovation on Scalable Data Access, with eight years funding, from the Research Council of Norway. He has been prominently involved in many national and European research projects; in particular, he was the Coordinator of the EU FP7 Project Envisage on formal methods for cloud computing, from 2013 to 2016, and the Scientific Coordinator of the EU H2020 Project HyVar on hybrid variability systems, from 2015 to 2018. His research interests include programming models and methodology, program specification and modeling, formal methods and associated theory, lightweight analysis, type systems, testing, and deductive verification and formal logic.

He is a member of IFIP WG2.2 "Formal Description of Programming Concepts". He was a Board Member of SINTEF ICT, from 2009 to 2015. He is currently a member of the Scientific Council of the Science Centre, UiO, a Board Member of the Formal Methods Europe, and a Steering Committee Member of the conference series on Fundamental Approaches to Software Engineering (FASE), Integrated Formal Methods (iFM), and Formal Techniques for Networked and Distributed Systems (FORTE). He was the General Chair of FM 2015 and DisCoTec 2008, and the PC Chair of FASE 2022, SEFM 2018, TAP 2017, ESOCC 2016, iFM 2013, and FMOODS 2007. He is an Editorial Board Member of the Journals *Formal Aspects of Computing* and *Journal of Logical and Algebraic Methods in Programming.*

ANDERS ANDERSEN is the Head of Department at the Department of Computer Science, UiT The Arctic University of Norway. He is Leading a national workgroup for the strengthening of ICT-security in technology studies in Norway, from 2019 to 2020. The workgroup is appointed by the Norwegian Association of Higher Education Institutions (UHR) on behalf of the Ministry of Education and Research.

The research of Anders Andersen covers four main domains. The first domain is security, where the focus is adaptable security, secure storage and sharing of data, security related to mobile systems, NFC and secure elements (e.g. SIM), and analysis of sensitive data (e.g. person-sensitive health data). The second domain is support for mobility and context, where configuration and reconfiguration of systems based on the current context are made possible with adaptable architectures. This domain includes integration of a wide range of services and information sources for the development of personalized and context sensitive solutions. The third domain is support for multimedia applications, including support for continuous media. He has used formal specifications directly for quality of service (QoS) management in running systems and he has developed an explicit binding architecture for multimedia communication. The fourth domain is adaptive system architectures, where he has developed programming abstractions for adaption control and techniques to observe and analyse system behavior.

His research interests include security, mobile services, personalisation, complex distributed applications, and privacy aware analysis of sensitive data.

ANDREA MARINONI (Senior Member, IEEE) received the B.S., M.Sc., *(cum laude)* and Ph.D. degrees in electronic engineering from the University of Pavia, Pavia, Italy, in 2005, 2007, and 2011, respectively.

From 2013 to 2018, he has been a Research Fellow at the Telecommunications and Remote Sensing Laboratory, Department of Electrical, Computer, and Biomedical Engineering, University of Pavia. In 2009, he has been a Visiting Researcher at the Communications Systems Laboratory, Department of Electrical Engineering, University of California–Los Angeles (UCLA), Los Angeles, CA, USA. In 2011, he has been the recipient of the two-year ''Applied Research Grant'', sponsored by the Region of Lombardy, Italy, and STMicroelectronics N.V. In 2017, he has been the recipient of the INROAD grant, sponsored by University of Pavia and Fondazione Cariplo, Italy, for supporting excellence in design of ERC proposal. In 2018, he has been the recipient of the ''Progetto Professionalità Ivano Becchi'' grant funded by Fondazione Banco del Monte di Lombardia, Italy, and sponsored by the University of Pavia and NASA Jet Propulsion Laboratory, Pasadena, CA, for supporting the development of advanced methods of air pollution analysis by remote sensing data investigation. He has been the recipient of Åsgard Research Programme and Åsgard Recherche+Programme grants funded by the Institut Français de Norvège, Oslo, Norway, in 2019 and 2020, respectively, for supporting the development of scientific collaborations between French and Norwegian research institutes. From 2015 to 2017, he has been a Visiting Researcher at the Earth and Planetary Image Facility, Ben-Gurion University of the Negev, Be'er Sheva, Israel; School of Geography and Planning, Sun Yat-sen University, Guangzhou, China; School of Computer Science, Fudan University, Shanghai, China.; Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, Beijing, China.; Instituto de Telecomunicações, Instituto Superior Tecnico, Universidade de Lisboa, Lisbon, Portugal. In 2020 and 2021, he has been a Visiting Professor at the Department of Electrical, Computer, and Biomedical Engineering, University of Pavia. He is currently an Associate Professor with the Earth Observation Group, Centre for Integrated Remote Sensing and Forecasting for Arctic Operations (CIRFA), Department of Physics and Technology, UiT the Arctic University of Norway, Tromsø, Norway, and a Visiting Academic Fellow with the Department of Engineering, University of Cambridge, U.K. His main research interests include efficient information extraction from multimodal remote sensing, nonlinear signal processing applied to large scale heterogeneous records, earth observation interpretation and big data mining, and analysis and management for human–environment interaction assessment.

He is the Founder and the Current Chair of the IEEE GRSS Norway Chapter. He is also an Ambassador for IEEE Region 8 Humanitarian Activities, and a Research Contact Point for the Norwegian Artificial Intelligence Research Consortium (NORA–nora.ai). He serves as a Topical Associate Editor of machine learning for IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING. He has been the Guest Editor of three special issues on multimodal remote sensing and sustainable development for IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING. He is also the leader of the GR4S committee within IEEE GRSS, coordinating the organization of schools and workshops sponsored by IEEE GRSS worldwide.