Towards On-device Machine Learning: Efficient Inference, Independent Learning, and Collaborative Unsupervised Learning

by

Yawen Wu

B.E., Shandong University, 2013

M.S., Shandong University, 2016

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2023

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Yawen Wu

It was defended on

Feb. 17th 2023

and approved by

Jingtong Hu, Ph.D., Associate Professor, Department of Electrical and Computer Engineering

Heng Huang, Ph.D., Professor, Department of Electrical and Computer Engineering

Zhi-Hong Mao, Ph.D., Professor, Department of Electrical and Computer Engineering

In Hee Lee, Ph.D., Assistant Professor, Department of Electrical and Computer Engineering

Xulong Tang, Ph.D., Assistant Professor, Department of Computer Science

Yiyu Shi, Ph.D., Professor, Department of Computer Science and Engineering, University of Notre Dame

Fei Fang, Ph.D., Assistant Professor, School of Computer Science, Carnegie Mellon University

Dissertation Director: Jingtong Hu, Ph.D., Associate Professor, Department of Electrical and Computer Engineering

Towards On-device Machine Learning: Efficient Inference, Independent Learning, and Collaborative Unsupervised Learning

Yawen Wu, PhD

University of Pittsburgh, 2023

With the increasing ubiquity of edge devices, such as the Internet of Things (IoT) and mobile devices, deploying machine learning models, particularly deep neural networks (DNNs), on these devices to extract information from sensed data can enable the democratization of artificial intelligence (AI). On-device AI has the potential to support various tasks, including wildlife monitoring, augmented reality, and autonomous driving.

To enable on-device AI, both on-device inference and training need to be achieved. On-device inference enables edge devices to make predictions from collected data. On-device training enables the devices to adapt to dynamic environments by learning from the environments and updating the model in situ. By applying on-device training to distributed devices, collaborative learning uses multiple devices to learn a shared model while keeping the data on personal devices for privacy.

However, it is challenging to achieve on-device inference and training. First, edge devices have limited computation capabilities and limited memory size, but DNNs are demanding in computation and memory. Therefore, DNNs need to be effectively compressed before deploying to edge devices. Second, there is a large gap between the high computation and energy demand of on-device training and the limited computing resources and battery capacity on edge devices. Third, during on-device training, each device can only collect a limited amount of data. However, model training needs a large amount of data to achieve a high generalization performance.

To address these challenges, this dissertation proposes several techniques to enable inference and training on single devices, and collaborative learning with multiple devices. First, a model compression framework is proposed to compress multi-exit neural networks by pruning and quantization. After compression, computation cost and model size are reduced while the accuracy is preserved. Second, an efficient training method is proposed to reduce the

computation cost of training by skipping unnecessary training data and pruning the gradient computation. To further learn with as few labels as possible, a data selection approach to select the representative data for training without using labels is proposed. Third, a collaborative unsupervised learning framework for distributed devices to learn a shared model from decentralized unlabeled data is proposed.

Table of Contents

List of Tables

List of Figures

Preface

First and foremost, I would like to sincerely thank my Ph.D. advisor, Professor Jingtong Hu, for his continuous guidance, support, advice, inspiration, enthusiasm, and patience with my Ph.D. study and research. He is an outstanding researcher and an exceptional instructor. He provides me with the opportunity to work with him. He teaches me how to look for and identify important research problems, form novel ideas to solve these problems, overcome difficulties in finding solutions to challenging problems, implement the solutions systematically and in-depth, and deliver the ideas and findings in papers and presentations. I am fortunate and it is my great hour to have the opportunity to work with Professor Jingtong Hu.

I would like to thank Professor Heng Huang, Professor Zhi-Hong Mao, Professor In Hee Lee, Professor Xulong Tang, Professor Yiyu Shi, and Professor Fei Fang, for serving as my Ph.D. dissertation committee members. They provide insightful advice and constructive feedback to my research and dissertation. I thank them for their valuable time and efforts in helping me.

I would like to thank all my academic collaborators. I would like to thank Professor Yiyu Shi for his inspiring ideas and guidance in almost all my research works. Without his help, feedback, and encouragement, some of my ideas would have been abandoned without further exploration when some initial solutions did not work. Thanks to his help, I managed to overcome the difficulties in developing and correcting ideas and had the opportunity to deliver the findings as research papers. I would like to thank Professor Fei Fang. I appreciate her great patience, insightful guidance, detailed suggestions, and constructive feedback for my first research paper. I also want to thank Professor In Hee Lee for his insightful suggestions. I also want to thank Professor Xulong Tang for his inspiring feedback on my research work. I also want to thank Professor Meng Li for his constructive suggestions for my research works. I also want to thank Professor Peipei Zhou for the great support of my research works. I also want to thank Professor Alaina J. James for the insightful suggestions from medical perspectives.

I want to thank all my lab mates, Dr. Mimi Xie, Dr. Chen Pan, Dr. Lei Yang, Dr.

Weiwen Jiang, Dr. Xinyi Zhang, Dr. Zhenge Jia, Dr. Xiaowei Xu, Zhepeng Wang, Dewen Zeng, Yue Tang, Jianing Deng, Pan Wang, Sheng Li, Yi Sheng, Yubo Du, Yuyang Li, Jinming Zhuang, and Zhuoping Yang for their support and help during my Ph.D. study. I want to especially thank my collaborators Zhepeng Wang and Dewen Zeng for their efforts in helping with my multiple research works.

Last but not least, I would like to thank my parents, my grandparents, my uncles, my aunts, and my cousins Guowen, Huijuan, and Huiting for their continuous support of me. Finally, I would like to thank my wife, Dr. Chaoqun Dong. Your love and support since high school give me encouragement and impetus for our wonderful and beautiful life journey. I also want to thank my parents-in-law for raising such an incredible and sweet person.

1.0 Introduction

Machine learning models, specifically deep neural networks (DNNs), have demonstrated remarkable performance in visual data analysis [23, 25, 36, 58, 97, 98, 105, 123], enabling sophisticated inference even with limited and noisy inputs. As edge devices such as IoT devices and mobile devices become ubiquitous nowadays, deploying machine learning models on such devices to extract information from the sensed data can enable us to democratize artificial intelligence (AI). On-device AI has the potential to power various tasks in our lives, from personal daily life assistance in smartphones, augmented reality (AR) and mixed reality (XR) glasses, wearable devices, medical devices [12, 116], smart cities [96], and smart agriculture [126], to search and rescue in Unmanned Aerial Vehicles (UAVs) [88, 91].

To enable on-device AI, both on-device inference and on-device learning need to be achieved. On-device inference enables edge devices to make predictions from collected data such as object classification to obtain the category of objects in the collected images, and object detection to recognize and predict bounding boxes of objects. Other than image data, data in other modalities can also be processed by on-device AI such as voice recognition. When these inference tasks are performed on high-performance computers (HPCs) with abundant resources, low latency, and high throughput can be easily achieved. However, it is not always possible to send all data collected on edge devices to HPCs for inference due to data privacy, communication cost, and communication latency concerns [13]. Therefore, it is desired to make predictions in situ on edge devices and only send the prediction results to HPCs for further processing. Compared with cloud servers, edge devices have limited computing capability and limited memory storage, which makes it challenging to perform inference on edge devices.

To address the challenge of on-device inference, researchers have developed techniques for efficient inference on edge devices. The multi-exit network with side classifiers in shallower layers is proposed [40, 99]. Instead of only having one final inference result, networks can have early results to save time or energy. Given an input, a subset of networks is selected for faster inference by trading off accuracy. A large portion of samples can exit the network

via these exits when the results at these exits have high confidence. Model compression methods including pruning and quantization are developed. [30, 37, 63] proposed pruning techniques to remove insignificant model parameters for smaller model sizes and faster inference. [44, 83, 102, 120, 125] quantize the model parameters and intermediate activations such that the high-cost floating-point operations are replaced by faster integer operations for speeding up inference.

In addition to on-device inference, on-device training is another technique needed to enable on-device AI. The deep learning models are usually trained on HPCs and then deployed to edge devices for inference. One drawback of existing deployment, however, is that typically they are "one size fits all". Once trained in the cloud and deployed in the devices, the neural networks do not adapt to different users and application domains, nor do they evolve when new unseen data streams in. However, in the physical world, the statically trained model cannot adapt to the real world dynamically and may result in low accuracy for new input instances. On-device training enables incremental and lifelong learning [85] to train an existing model to update its knowledge. It also enables the deployment of transfer learning and personalization [1–3] to improve accuracy for different users and application scenarios. These techniques also continuously improve the performance of people that were under-represented during the data collection stage. Federated learning [74] is another application scenario of on-device training, where a large number of devices (typically mobile phones) collaboratively learn a shared model while keeping the training data on personal devices to protect privacy. The challenge of on-device training is the large gap between the high computation and energy demand of training and the limited computing resource and battery on edge devices. To enable on-device training, the computation cost of training needs to be significantly reduced while the accuracy is preserved.

1.1 Challenges of On-device Machine Learning

A key requirement for an independent on-device machine learning system is to achieve accurate and efficient predictions. To this end, the efficient deployment of machine learning

models for inference, the capacity to learn from new data and adapt to the dynamic environment, and the ability to share knowledge among devices are crucial components for on-device machine learning. This dissertation addresses three main challenges associated with building independent on-device learning systems.

1.1.1 Limited On-device Resources

Edge devices typically have constrained computing capabilities and limited memory size. While DNNs can effectively extract features from noisy input data, they are usually computationally expensive and require large memory sizes to make predictions. Typical neural networks have tens of millions of weights and use billions of operations to finish one inference. Even a small DNN (e.g. MobileNetV2 [89]) has over a million weights and millions of operations. However, edge devices such as microcontrollers (MCUs) are constrained in resources. Typical MCUs have limited storage (e.g. Flash or FRAM) size (several or tens of KB) and run in low frequency (several or tens of MHz). Directly deploying conventional DNN to MCU is infeasible because the model size exceeds the storage capacity. Even if the DNN model can fit into the limited storage, the latency to finish one inference is still too long. For other edge devices such as the mobile GPU platform Nvidia Jetson [78], while the memory size is large enough for DNNs, it is necessary to reduce the computation cost of DNNs to reduce the latency of inference, save energy and prolong the battery life. The recently developed multi-exit neural networks [99] can reduce the inference latency by exiting from side classifiers. However, their model sizes are even larger than conventional DNNs due to the additional side classifiers. As a result, in order to deploy DNNs on edge devices, the computation cost and memory footprint of DNN models need to be effectively reduced.

1.1.2 High Training Costs and Label Scarcity

Typically, DNN models are trained on high-performance computers and then deployed to edge devices. However, these statically trained models are unable to adapt to dynamic real-world environments, which can result in low accuracy for new inputs. On-device training by learning from real-world data after deployment can significantly improve accuracy. Unfor-

tunately, the high computation cost of training can be prohibitive for resource-constrained devices, posing a significant challenge for on-device learning. The challenge is amplified by the large gap between the high computation and energy demand of training and the limited computing resources and battery capacity of edge devices. For example, training ResNet-110 [35] on a 32x32 input image takes 780M FLOPs, which is prohibitive for IoT devices. Besides, since computation directly translates into energy consumption and IoT devices are usually battery-constrained [30], the high computation demand of training will quickly drain the battery. While existing works [45, 80, 108] effectively reduce the computation cost of inference by assigning input instances to different classifiers according to the difficulty, the computation cost of training is not reduced. Therefore, a method to reduce the training cost to enable on-device training is needed.

Besides, during on-device training, the data are collected from the input stream and are unlabeled. To perform on-device training, while it is feasible to send a few data to servers for labeling, it is prohibitive to send all these new data due to the requirement of expert knowledge, data privacy, communication cost, and latency concerns [13]. Thus, it is also needed to learn from new streaming data in-situ with as few labels as possible.

1.1.3 Limited Data on Individual Devices

During on-device training, each device can only collect a limited amount of data. Since effective model training needs a large amount of data, training on limited data can result in model overfitting and degrade the model's generalization performance [22]. Collaborative learning is an effective approach for multiple distributed devices to collaboratively learn a shared model from decentralized data and avoids raw data sharing for privacy protection. However, data collected on devices are inherently far from independent and identically distributed (IID) [39], which results in ineffective collaborative learning, especially when most of the data are unlabeled. To improve the performance of collaborative learning on non-IID data, [46, 124] share local raw data (e.g. images) among clients. However, sharing raw data among clients will cause privacy concerns [64]. Besides, they need fully labeled data to perform collaborative learning, which requires expert knowledge and potentially high

labeling costs. Therefore, an approach to performing collaborative learning with limited labels and avoiding sharing raw data is needed.

1.2 Research Overview

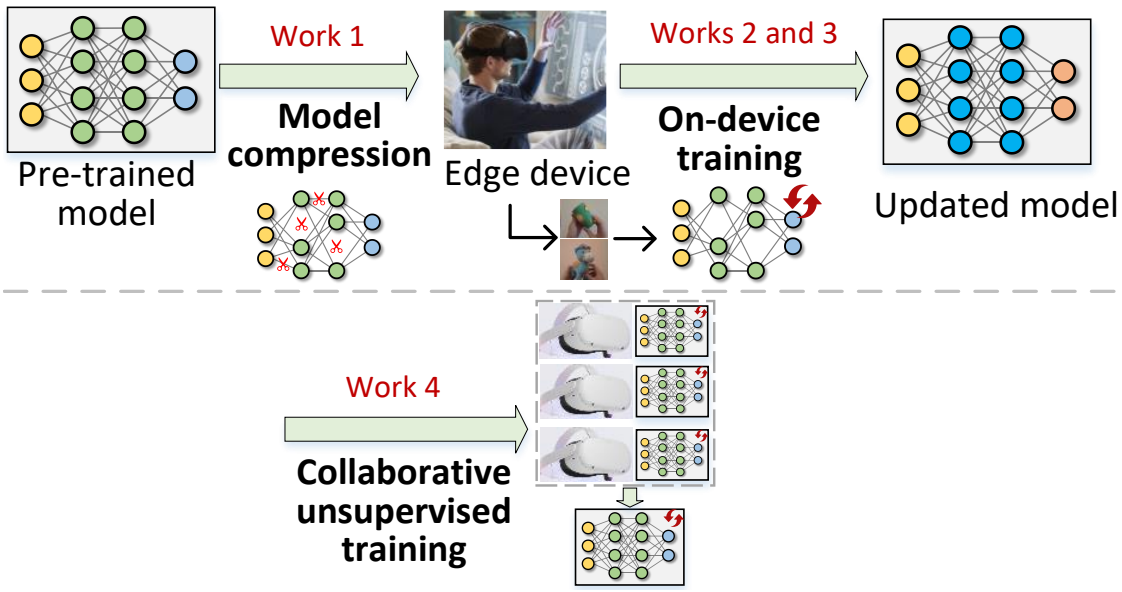


Figure 1: Overview of the proposed approaches in the dissertation.

The overview of this dissertation is shown in Figure 1. A framework for on-device machine learning is proposed. In this framework, there are three major steps to achieve on-device machine learning:

1. Model compression of multi-exit neural networks for efficient on-device inference.
2. On-device training on single devices with labeled or unlabeled data.
3. Collaborative on-device training among devices.

As discussed in Section 1.1.1, edge devices usually have limited computation capabilities and limited memory size. Since DNNs are computationally intensive and have a large memory footprint, directly deploying pre-trained models to devices is prohibitive. To address this

challenge, as shown in Work 1 of Figure 1, we propose a model compression framework to effectively compress a multi-exit neural network by model pruning and quantization with a nonuniform compression ratio and quantization bit-width. After compression, computation cost and model size are reduced while model accuracy is preserved [108]. The proposed model compression method is illustrated in the energy harvesting scenario, but it is generally applicable to edge devices.

As discussed in Section 1.1.2, the environment in the physical world is dynamic. When using a statically trained model, it cannot adapt to the real world dynamically and may result in low accuracy for new input instances. On-device training by learning from real-world data after deployment can greatly improve accuracy. However, the high computation cost makes training prohibitive for resource-constrained devices. To enable on-device training, Works 2 and 3 in Figure 1 are proposed, In Work 2, we explore the computational redundancies in training and propose a framework to reduce the computation cost by skipping unnecessary training data and pruning the gradient computation [109]. To further learn data in situ with as few labels as possible, in Work 3, a data selection approach to select the most representative data from the input data stream without using labels is proposed [111].

As discussed in Section 1.1.3, each device can only collect a limited amount of data and may not train the model well with limited data. To address this challenge, we propose a decentralized unsupervised learning framework for multiple distributed clients to collaboratively learn a shared model from decentralized data [110]. In this framework, feature fusion is proposed to solve the problems of imbalanced data distribution among devices, and neighborhood matching is proposed to learn a unified feature space among devices. In addition to computer vision tasks on edge devices, this framework can also be applied to general medical image segmentation problems [113–115].

1.3 Contributions

Research contributions for this dissertation can be concluded as follows:

1.3.1 Model Compression for Multi-exit Neural Networks

We aim to enhance the efficiency of on-device inference by utilizing compressed multi-exit neural networks [108]. More specifically, the proposed model compression method makes the following contributions:

- **Model compression framework for multi-exit neural networks.** We propose a framework for pruning and quantizing multi-exit neural networks to fit onto MCUs while maximizing the average inference accuracy.
- **Application of compressed multi-exit neural networks for energy harvesting scenarios.** We apply the compression framework to energy harvesting (EH) applications. Based on the multi-exit neural network, we introduce an intermittent and incremental inference model to guarantee an inference result before power failure occurs. We develop a power trace-aware and exits-guided compression technique. We also propose an online exit selection method, which selects the exit for each event based on the EH condition and difficulty of each input.

1.3.2 Efficient Supervised On-device Training

This work aims to enable on-device training of convolutional neural networks (CNNs) by reducing the computation cost at training time [109]. To be specific, the contributions of this work are as follows:

- **A framework to enable on-device training.** We propose a framework consisting of two approaches to eliminate unnecessary computation in training CNNs while preserving full network accuracy. The first approach improves the training efficiency of both the forward and backward passes, and the second approach further reduces the computation cost in the backward pass.

- **Self-supervised early instance filtering (EIF) on the data level.** We propose an instance filter to predict the loss of each instance and develop a self-supervised algorithm to train the filter. Instances with predicted low loss are dropped before starting the training cycle to save computation. To train the filter simultaneously with the main network, we propose a self-supervised training algorithm including the adaptive threshold-based labeling strategy, uncertainty sampling-based instance selection algorithm, and weighted loss for biased high-loss ratio.
- **Error map pruning (EMP) on the algorithm level.** We propose an algorithm to prune insignificant channels in error maps to reduce the computation cost in the backward pass. The channel selection strategy considers the importance of each channel on both the error propagation and the computation of the weight gradients to minimize the influence of pruning on the final accuracy.

1.3.3 Self-supervised On-device Training

This work aims to enable on-device contrastive learning from input streaming data by selecting the most representative data [111]. The contributions are as follows.

- **Self-supervised on-device learning framework.** We propose a framework for self-supervised contrastive learning that forms mini-batches of training data on-the-fly from the unlabeled input stream. The framework leverages a small data buffer and eliminates the need to store all the streaming data on the device.
- **Contrast scoring for data selection.** We propose a data replacement policy by contrast scoring to maintain the most representative data in the buffer for on-device contrastive learning. Labels are not needed in the data replacement process, and the selected data will generate large gradients that benefit the learning most.
- **Lazy scoring for reduced computation overhead.** We propose a lazy scoring strategy to reduce the runtime overhead of data scoring. The data scores are updated every several iterations instead of every iteration to save computation.

1.3.4 Collaborative Unsupervised Learning with Decentralized Devices

A collaborative unsupervised learning method is proposed to enable distributed devices for learning a shared model from decentralized unlabeled data by feature fusion and neighborhood matching [110]. The contributions are as follows.

- **Collaborative contrastive learning framework.** We propose a framework with two approaches to learning visual representations from unlabeled data on distributed clients. The first approach improves the local representation learning on each client with limited data diversity, and the second approach further learns unified global representations among clients.
- **Feature fusion for better local representations.** We propose a feature fusion approach to leverage remote features for better local learning while avoiding raw data sharing. The remote features serve as negatives in the local contrastive loss to achieve a more accurate contrast with fewer false negatives and more diverse negatives.
- **Neighborhood matching for improved global representations.** We propose a neighborhood matching approach to cluster decentralized data across clients. During local learning, each client identifies the remote features to cluster local data with and performs clustering. In this way, well-clustered features among clients can be learned.

1.4 Dissertation Organization

The dissertation is organized as follows. Chapter 2 introduces a model compression framework to compress multi-exit neural networks by model pruning and quantization for dynamic on-device inference. Chapter 3 presents an efficient supervised on-device training framework for reducing the training cost while preserving the desired accuracy. Chapter 4 presents an on-device self-supervised learning framework to learn from the unlabeled input stream by selecting the most representative data. Chapter 5 introduces a decentralized unsupervised learning framework, which enables distributed clients to collaboratively learn a shared model from decentralized unlabeled data. Chapter 6 summarizes this dissertation.

2.0 Efficient On-device Inference with Compressed Multi-exit Neural Networks

This chapter presents a project that aims to improve the efficiency of on-device inference by compressing multi-exit neural networks [108]. The chapter is organized as follows. First, the background and the motivation are introduced, and the related works are discussed. Next, the details of the proposed model compression method for multi-exit neural networks are presented. Then, a case study of applying the proposed model compression technique to energy-harvesting powered event-driven IoT systems is introduced. After that, the experimental results are shown to demonstrate the effectiveness of the proposed techniques, and a video demo is described. Finally, the conclusion is presented to summarize this project.

2.1 Introduction

DNNs have achieved record-breaking performance for visual data analysis and enable sophisticated inference using limited and noisy inputs. DNNs can effectively extract higher-level feature representations from noisy input data. As edge devices such as IoT devices and mobile devices become ubiquitous nowadays, deploying DNNs on such devices to extract information from the sensed data can enable us to democratize artificial intelligence (AI).

However, DNNs are usually computationally expensive. Typical neural networks have tens of millions of weights and use billions of operations to finish one inference. Even a small DNN (e.g. MobileNetV2 [89]) has over a million weights and millions of operations. However, MCUs are constrained in resources. Typical MCUs have limited storage (e.g. Flash or FRAM) size (several or tens of KB) and run in low frequency (several or tens of MHz). Directly deploying conventional DNN to MCU is infeasible because the model size exceeds the storage capacity. Even if the DNN model can fit into the limited storage, the latency to finish one inference is still too long. For other edge devices such as mobile GPU Nvidia Jetson [78], while the memory size is large enough for DNNs, it is necessary to reduce the computation cost of DNNs to reduce the latency of inference, save energy and prolong the

battery life.

To reduce the latency and energy usage of DNN inference, the multi-exit neural network with side classifiers in shallower layers is proposed [40,99]. By using this model architecture, a large portion of samples that can have high confidence in predictions at the shallower classifier can exit the network early. For more difficult samples, more network layers or the full network will be used to provide a more accurate result [99]. Multi-exit neural networks are very promising for edge devices with limited resources because they can reduce the inference energy cost and latency by exiting from early exits while maintaining prediction accuracy. While multi-exit neural networks can reduce the inference latency by exiting from side classifiers, the network size with multi-exits is usually larger than single-exit ones in terms of weights and intermediate feature maps due to the additional layers to capture high-level features along each exit path. As a result, to deploy multi-exit neural networks to edge devices, the computation cost and memory footprint need to be effectively reduced.

To achieve efficient inference with multi-exit networks on edge devices, the challenge is how to fit the multi-exit network onto MCUs while keeping a high accuracy of each exit. Simply compressing the network with existing network compression approaches [27] does not work well since they only consider the accuracy of the final exit. For a multi-exit network, only considering the final exit during compression will significantly degrade the accuracy of early exits. Unfortunately, in general applications, a large portion of the samples are easy ones [100]. Early exits in shallower layers are often chosen to generate the result with a limited energy budget, which results in low accuracy. Therefore, how to compress the network considering the accuracy and energy cost of each exit remains a problem.

To address this challenge, we propose a framework to automatically compress multi-exit neural networks before deployment. We aim to compress the multi-exit network to fit it onto edge devices and achieve a high average accuracy of all samples. First, we will consider the typical data distribution of the target application, which determines the probability of selecting each exit. Priority will be given to the exits which have a higher probability of being selected. Since the probability of selecting each exit will change after we compress the network due to the change of prediction capability for each exit, we develop a reinforcement-learning (RL) based approach to automatically search the best pruning rate, bitwidth of weights and

activations in all the layers to maximize the average accuracy.

One of the most promising applications of multi-exit neural networks on edge devices is to build persistent, event-driven IoT systems. Recently, the maturation of energy harvesting (EH) technology and the emergence of intermittent computing, which stores harvested energy in energy storage and supports an episode of program execution during each power cycle, creates the opportunity to build sophisticated battery-less energy-neutral sensors. Such sensors have the potential to build persistent sensing systems, in which the main device (e.g. a battery-draining processing device) can remain dormant, with near-zero power consumption, until awakened by an EH-powered sensor, which monitors events of interest constantly with harvested energy. To realize this capability, the EH-powered sensor has to frequently make decisions locally with sensor data, as it is prohibitive to send the raw data to other devices and offload the computation to them.

DNN inference on intermittently-powered devices remains largely unexplored. Existing work [27] made the first step to implement DNNs on an intermittently powered MCU. However, multiple power cycles are needed to finish one inference in most cases. Since the harvested power is usually weak and unpredictable, the latency to obtain the final inference result can be indefinitely long. Multi-exit neural networks have the potential for efficient and timely inference on EH-powered devices. Based on the available energy, an exit can be selected such that an inference result can be obtained before power failure. This eliminates the need to wait for multiple power cycles to finish one inference and generates a timely result.

However, to compress multi-exit neural networks for EH-powered devices, there are two challenges. First, when powered by EH, in addition to considering multi exits during compression, the EH condition also needs to be considered. Powered by dynamic EH, the chances that each exit is selected are different depending on both the power condition and the accuracy/energy cost of each exit after compression. To maximize the average accuracy of all the events, the compression algorithm has to take the power condition and accuracy/energy cost of multiple exits into consideration. Maximizing the average accuracy across all the events is equivalent to maximizing the number of interesting events that are correctly processed in a fixed amount of harvested energy, which is important for EH-powered devices.

The second challenge is how to select the exit for each event during runtime to achieve

high average accuracy in the long term. The exit needs to be selected based on the available EH energy and the difficulty of each input. Two sequential decisions need to be made. First, when an event happens, simply selecting the exit with the highest accuracy that current energy can support can result in low average accuracy in the long run. This is because even if the current EH efficiency is high, it can be low in the future. Instead of using up all the available energy for one inference to achieve high accuracy, a better strategy is to reserve some energy for future events. Otherwise, the following events will have low accuracy or even be missed because of insufficient energy. Second, the inference difficulty of each input needs to be considered. The difficulty is only known at an exit by inspecting the entropy of the current result. If the confidence is low at the selected exit, a second decision needs to be made on whether an incremental inference is needed to propagate the input to a following exit for higher accuracy. To make these two decisions, the EH condition and the difficulty of the current event need to be considered.

To address these two challenges, based on the proposed compression framework for multi-exit networks, we further propose a two-phase approach to compress multi-exit networks for EH devices and conduct runtime exit selection. In the first phase, we apply the compression framework and add constraints of typical EH power traces and event distribution to determine the probability of selecting each exit. After compression, a pruned and quantized neural network that maximizes the average accuracy of the typical power trace and event distribution is obtained.

In the second phase, we aim to maximize the average accuracy for all the events during runtime. We employ Q-learning [106] to learn the best exit under different EH energy conditions. To select the exit for an event, we use the currently available energy level and the charging efficiency as the state, and use all exits as the actions the learning method can take. Q-learning is lightweight as it uses a lookup table (LUT) to select actions. The learning process only involves updating the LUT. To decide whether to conduct incremental inference, we use the confidence of the result at the selected exit and currently available energy as the state. The action is a binary decision, representing to continue the inference or to output the current result.

In summary, the main contributions of the work include:

- **Compression framework for multi-exit neural networks.** We propose a framework to prune and quantize multi-exit neural networks to fit onto MCUs while maximizing the average inference accuracy.
- **Application of compressed multi-exit neural networks for energy harvesting scenarios.** Based on the multi-exit neural network, we propose an intermittent and incremental inference model to guarantee an inference result before power failure occurs. We apply the compression framework to EH applications and develop a power trace-aware and exits-guided compression technique. We propose an online exit selection method to select the exit for each event considering the EH condition and difficulty of each input.

Experimental results of applying the proposed model compression framework to the EH application scenario show that this compression framework improves the number of correctly processed events per energy unit by 3.6x over [27], a state-of-the-art (SOTA) intermittent inference framework. It also outperforms [24], a NAS framework to generate networks for MCUs, by 18.9x. The latency of all the processed events is improved by 7.8x and 10.2x over these two approaches, respectively.

2.2 Related Work

2.2.1 Multi-exit Network

The multi-exit neural network has been investigated in various studies. Instead of only having one final inference result, networks can have early results to save time or energy. In [40, 99], a subset of networks is selected for faster inference by trading off accuracy. These approaches allow a dynamic trade-off between inference latency and accuracy. However, none of the works are targeted at EH-powered MCUs, which are constrained in the weight storage and energy budgets. The large weight size and FLOPs of their models are prohibitive for direct deployment to EH-powered MCUs. Pruning and quantization are needed for the deployment.

2.2.2 Network Compression

There are extensive explorations on network pruning and quantization. For quantization, [83] employs binary filters and inputs for CNNs. [102] automates the quantization of each layer by a learning-based method. For pruning, [37] employs RL to automatically explore the layer-wise pruning rate for channel pruning [63]. However, these pruning and quantization methods only consider networks with one exit, which will greatly degrade the accuracy of early exits. Besides, the above approaches only focus on either quantization or pruning, not both of them. To deploy multi-exit networks to MCUs, an automated approach to conduct the quantization and pruning simultaneously while considering the accuracy of all exits is needed.

2.2.3 Intermittent Execution

EH techniques extract power from the ambient environment and provide an attractive power alternative in sensing scenarios where it is difficult to employ batteries. With an unstable power supply, EH-powered computing systems have to run intermittently. Various optimization and tools such as Chain [20] have also been proposed to ensure correctness and improve efficiency. Gobieski et al. [27] made the first step to implementing DNNs in an intermittently-powered sensor. It guarantees the correctness of DNN inference across multiple power failures. The drawback is that we must wait for multiple power cycles to finish one inference. Since the harvested power is usually weak and unpredictable, it takes an indefinite amount of time to obtain the final inference result.

2.3 Model Compression of Multi-exit Neural Networks

In this section, we will develop an exit-guided network compression framework. It aims to fit the multi-exit network onto edge devices and maximize the average accuracy by allocating layer-wise pruning rate and quantization bitwidth. Figure 2 shows a simple network with three exits, and each exit has a different accuracy and energy cost. Existing model compression

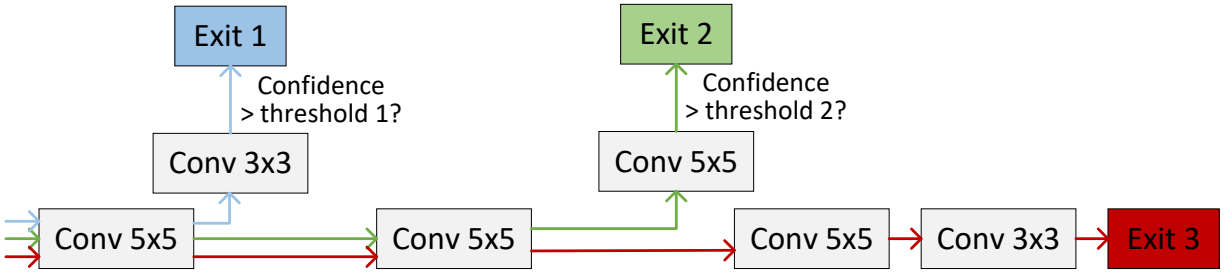


Figure 2: Architecture of multi-exit neural networks.

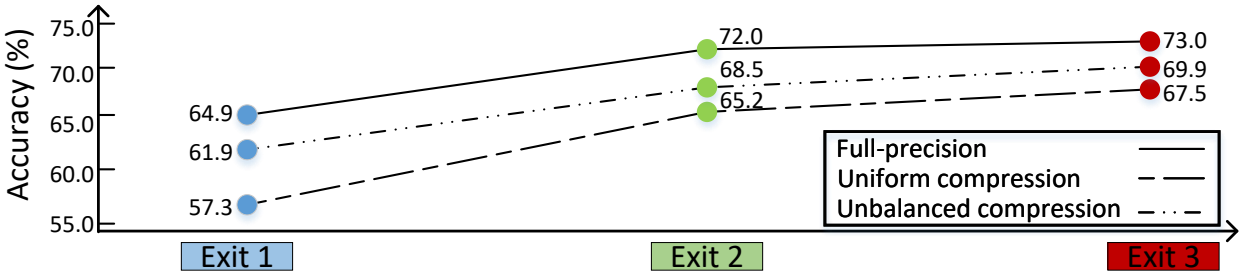


Figure 3: Benefits of nonuniform compression.

algorithms, which uniformly compress networks, will significantly degrade the accuracy of exits in shallow layers as shown in Figure 3. Different from existing algorithms that only consider the accuracy of the final classifier, the proposed framework takes the accuracy of all exits into consideration and conducts nonuniform compression. As shown in Figure 3, if we take a non-uniform approach, which compresses less in the shallow layers and more in the deep layers, the accuracy drop for all exits will be small. What is more, some exits will be chosen more often than others under a given data distribution. Thus, we will prioritize the accuracy of these exits during the compression process. In this way, we can improve the average inference accuracy across all data.

The overview of the compression approach is shown in Figure 4. This approach takes the multi-exit network and training samples as the input and generates a non-uniform pruning rate and the bitwidth allocation policy for each layer. Based on the pruning rate, channel

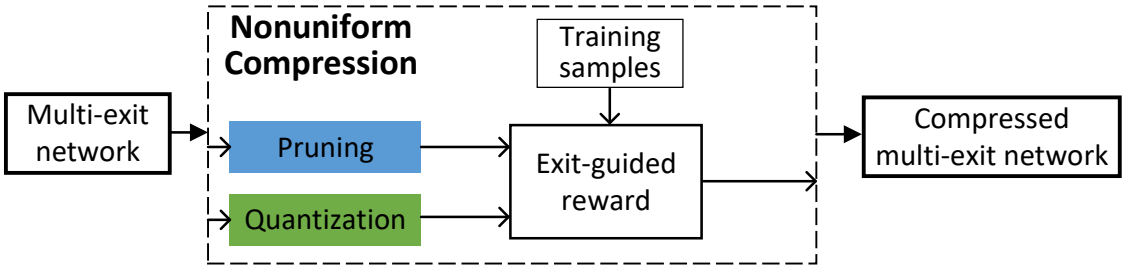


Figure 4: Overview of the proposed compression framework.

pruning is applied to each layer to prune out the input channels [63]. The channel to be pruned out is selected by the importance of the channel, i.e. the magnitude of weights applied to the input channel, and the less important ones are pruned out. Based on the bitwidth policy, linear quantization [102] is applied to both the weights and activations. After compression, the network is deployed onto edge devices and the exit will be selected by the measured entropy at each exit and the entropy threshold. During compression, the approach first generates an initial layer-wise compression policy. The compression policy prioritizes the exits with higher probability and provides them with relatively higher accuracy by adjusting the layer-wise compression policy. After applying the compression policy, the probability distribution of each exit is changed and the compression policy needs further fine-tuning. To accelerate the above iterative design process, we propose a reinforcement learning (RL)-based algorithm to co-explore the pruning and quantization policies and the probability distribution of each exit.

2.3.1 Problem Formulation

Given a full-precision network with multiple early exits, we will explore the accuracy and energy cost allocation for each exit to maximize the average accuracy under the given training data distribution. This is achieved by non-uniformly allocating the pruning rate and quantization bitwidth for each layer. Both pruning and quantization reduce the FLOPs and weight size of the network but with different emphases. Pruning mainly reduces the FLOPs,

while quantization mainly reduces the model size.

2.3.1.1 Pruning

Given a pruning rate α_l, we employ channel pruning to prune out the entire input channels of a convolutional or fully-connected layer. The advantages are two-fold. It reduces the FLOPs of the previous layer by reducing the number of output channels. It also reduces the FLOPs of the current layer by reducing the number of input channels. Besides, it can be directly implemented on off-the-shelf MCUs without the overhead. More specifically, given the pruning rate α_l for layer l, we reduce the filter weights from shape $[n, c, k, k]$ to $[n, c', k, k]$ such that $\alpha_l = c'/c$. For convolutional layers, n and c are the numbers of output and input channels, respectively, and k is the filter kernel size. For fully-connected layers, n and c are the numbers of output and input activations, and $k = 1$. The input channels to be pruned are selected according to the sum of absolute weights applied to them. We use $w_{i,j}$ to represent the weights of filter i connected to input channel j. The importance of input channel j is:

$$s_j = \sum_{i=1}^{n} |W_{i,j}|, \ j \in \{1, ..., c\}. \tag{2-1}$$

All the input channels are sorted by s_j and the least important ones are pruned out to make $c' = c$.

2.3.1.2 Quantization

For each layer l, we employ linear quantization for both the weights and activations following the bitwidth b_l^w and b_l^a. Given weight bitwidth $b_l^w = k$, the linearly quantized weight w_l' is:

$$w_l' = clamp(round(w_l/s), -2^{k-1}, 2^{k-1} - 1) \times s, \tag{2-2}$$

where $clamp(x, lb, ub)$ truncates the value x into the range $[lb, ub]$ that k bits can represent. s is the scaling factor, which is determined by minimizing the quantization error $||w_l' - w_l||_2$. As for activations, the quantization procedure is similar except the range for $clamp()$ is changed. Since all the activations are non-negative due to the ReLU function, we truncate

the activations into the range $[0, 2^k - 1]$.

The goal here is to find the best pruning and quantization rate. Formally, the multi-exit network compression problem is formulated as:

$$\text{Max } \frac{1}{N} \sum_{j \in \mathcal{D}} Acc_{exit(j)} \tag{2-3}$$

$$\text{s.t.} \quad Acc_i = f_{acc}(\alpha_1, b_1^w, b_1^a, ..., \alpha_{L_i}, b_{L_i}^w, b_{L_i}^a), \quad \forall i \in \{1...m\}, \tag{2-4}$$

$$S_{model} \leq S_{target}, \tag{2-5}$$

$$F_{model} \leq F_{target}. \tag{2-6}$$

The objective is to maximize the average accuracy for the given dataset $\mathcal{D}$ with N samples. In the objective function Eq.(2-3), $Acc_{exit(j)}$ represents the accuracy of the exit for data sample j. For sample j, an exit i is selected from m exits by the policy $i = exit(j)$. A simple policy is selecting the exit for a sample such that the confidence of the prediction is larger than a given confidence threshold. The first constraint listed in Eq.(2-4) is that the accuracy Acc_i of exit i is determined by the pruning rate α_l, weight bitwidth b_l^w and activation bitwidth b_l^a of all layers before the layer L_i where exit i is located. The second constraint listed in Eq.(2-5) is the weight size S_{model} can fit into the target MCU. The third constraint listed in Eq.(2-6) is that the total FLOPs F_{model} is reduced to the target value F_{target}.

Given the pruning rate α_l and bitwidth $b_l^w, b_l^a, l \in \{1, ..., L\}$, the objective function can be immediately calculated. This is done by first evaluating Eq.(2-4) on the representative dataset to get Acc_i. Following exit selection policy, the exit $i = exit(j)$ for sample $j \in \{1...N\}$ is determined. Given $exit(j)$, the objective function Eq.(2-3) is calculated. However, the search space is prohibitively large to find the optimal allocation policy. Assume the network has L layers. The quantization bitwidth b_l^w and b_l^a are both selected from $\{1, ..., 8\}$, and the pruning rate α_i is in the range [0.05,1.0] with a step size of 0.05. The design space as large as $(8^2 \times 20)^L \approx 10^{3L}$, which prohibits direct searching.

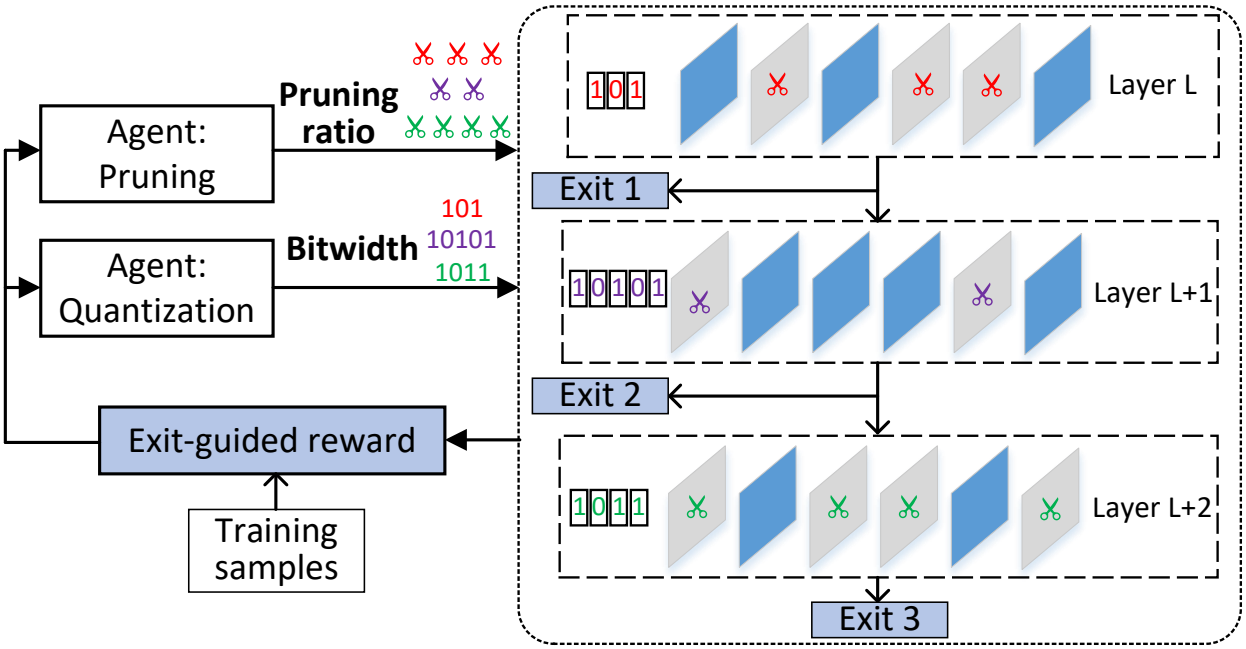


Figure 5: Exit-guided layer-wise pruning and quantization.

2.3.2 RL-Based Nonuniform Compression

To effectively search for the optimal parameters, we model the pruning and quantization task as a reinforcement learning problem. As shown in Figure 5, we use two agents to generate the pruning rate and quantization bitwidth layer-by-layer. The compressed network is then evaluated with the training data. the exit is selected according to the confidence threshold. After that, the reward representing the average accuracy of all samples is given to the agents to update their policies. After the exploration, the agents will generate the pruning rate and quantization bitwidth for each layer to maximize Eq.(2-3).

2.3.2.1 State

Two agents share the layer-wise state during training and generate different actions. The key point is that both the pruning and quantization information are encoded in the observation. Each agent observes the peer's action in the last layer such that it can take

action accordingly. For layer l, the shared observation O_l is:

$$O_l = (l, \alpha_{l-1}, b^w_{l-1}, b^a_{l-1}, flop_{reduced}, flop_{remain}, s_{reduced}, s_{remain}, i_{conv}, c_{in}, c_{out}, s_{weight}), \quad (2\text{-}7)$$

where l is the layer index, α_{l-1} is the pruning rate of the previous layer, and b^w_{l-1} and b^a_{l-1} are the bitwidth of weights and activations of the previous layer. $flop_{reduced}$ is the reduced FLOPs in previous layers, and $flop_{remain}$ is the FLOPs in the following layers. $s_{reduced}$ and s_{remain} are the reduced weight size and the remaining weight size. i_{conv} is a binary value indicating whether this layer is a convolutional or fully-connected layer. c_{in} and c_{out} are the number of input and output channels for the convolutional layer, or the number of input and output activations for the fully-connected layer. Each dimension of O_l is normalized to $[0, 1]$ to make them on the same scale.

2.3.2.2 Action

Two agents generate different actions. One agent generates the action α_l for the layer-wise pruning rate. The other agent generates two actions, one for the layer-wise weight bitwidth b^w_l and one for activation bitwidth b^a_l. We use continuous action space to generate accuracy pruning rate and quantization bitwidth. We do not use discrete action space because fine-grained pruning rate and quantization bitwidth need a large number of discrete actions to represent, which results in inefficient exploration during training. To apply the agents' actions to the compression process, the continuous action representing the pruning rate can be directly applied to pruning. The action for quantization is first linearly mapped from the continuous action space $[0, 1]$ to the discrete bitwidth in the range $[b^w_{min}, b^w_{max}]$ for weights and $[b^a_{min}, b^a_{max}]$ for activations. Then the bitwidth is applied to the quantization of weights and activations.

2.3.2.3 Reward

Two agents have different reward functions R_{prune} and R_{quant} due to different goals. Their rewards consist of the accuracy part R_{acc} and the compression part. R_{acc} aims to maximize the average accuracy of all samples. We use the percentage of each exit selected to guide the

compression process. R_{acc} is defined as:

$$R_{acc} = \sum_{i=1}^{m} p_i Acc_i, \tag{2-8}$$

where p_i is the percentage of exit i being selected. It is determined by the given data distribution and the exit selection policy such as the confidence threshold policy.

The compression goal of the pruning agent is to keep the FLOPs of all exits $F_{model} = \sum_{i=1}^{m} flop_i$ under the targeted value F_{target}. The quantization agent aims to keep the weight size S_{model} under the target value S_{target}. Considering the accuracy reward in Eq.(2-8), the reward for two agents is defined as follows:

$$R_{prune} = \begin{cases} \lambda_1 R_{acc} & \text{if } F_{model} \leq F_{target}, \\ -\lambda_1 & \text{otherwise,} \end{cases} \tag{2-9}$$

$$R_{quant} = \begin{cases} \lambda_2 R_{acc} & \text{if } S_{model} \leq S_{target}, \\ -\lambda_2 & \text{otherwise,} \end{cases} \tag{2-10}$$

where λ_1 and λ_2 are the reward scaling factors. When the compression goal is satisfied, the reward is the scaled accuracy. Otherwise, the reward is a negative value to punish the agents.

2.3.2.4 Agent

We use two RL agents, one for pruning and the other for quantization. Separate agents enable us to set different rewards to achieve different goals simultaneously. The agents leverage the deep deterministic policy gradient (DDPG) [65] algorithm to explore the design space. The agents process the network layer-by-layer. In the learning process, one step represents that the agent processes one layer. For each layer, two agents take the step simultaneously and proceed to the next layer. One episode consists of many steps. It starts from the first layer and ends at the last layer.

During exploration, each agent aims to maximize the overall reward of one episode. The action-value Q-function is estimated as

$$Q'_l = r_l + Q(O_{l+1}, a_{l+1})|_{a_{l+1}=\mu(O_{l+1})}. \tag{2-11}$$

The Q-function $Q(O, a)$ is updated by minimizing the loss:

$$Loss = \frac{1}{N} \sum_{l} (Q'_l - Q(O_l, a_l))|_{a_l=\mu(O_l)}, \tag{2-12}$$

where N is the number of sampled steps during exploration. The policy $a = \mu(O)$ is updated using the sampled policy gradient:

$$\nabla J = \frac{1}{N} \sum_{l} \nabla_{a_l} Q(O_l, a_l) \nabla \mu(O_l). \tag{2-13}$$

2.4 Application of Compressed Multi-exit Neural Network: Event-Driven IoT with Energy Harvesting

In the above sections, we have introduced the model compression of multi-exit neural networks for general scenarios. Next, we provide a case study of the proposed techniques in energy harvesting-powered event-driven IoT systems. In this system, the main device (e.g. a battery-draining processing device) can remain dormant, with near-zero power consumption, until awakened by an EH-powered sensor, which monitors events of interest constantly with harvested energy.

We first describe the intermittent inference model with multi-exit neural networks in Section 2.4.1. Then we introduce how to apply the proposed model compression framework to this EH application scenario in Section 2.4.2. Finally, we describe runtime exit selection in Section 2.4.3.

2.4.1 Event-Driven Intermittent Inference

2.4.1.1 Intermittent Execution Model

In the existing SOTA deployment of DNNs on EH-powered devices [27], when the power is not sufficient to finish the entire forward pass, the system is forced to pause during the inference process and wait until enough energy is harvested. However, the unpredictable EH process can result in indefinite waiting time to harvest sufficient energy, by which time the

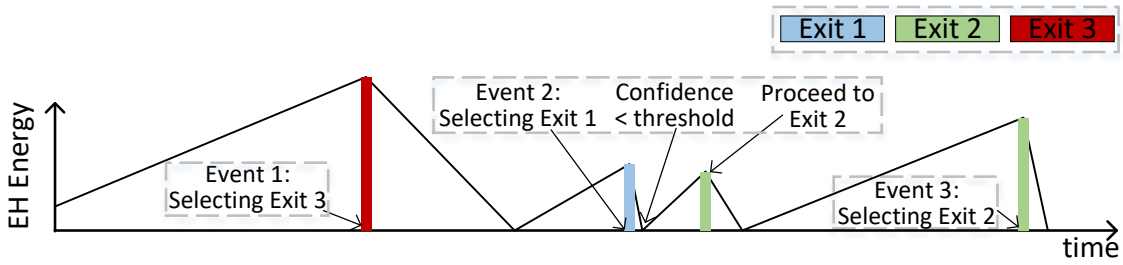


Figure 6: Intermittent execution model with multi-exits.

event may become obsolete. To solve this problem, we employ networks with multi-exits [40]. As shown in Figure 2 and Figure 3, this simple network has 3 exits, and each exit has a different accuracy and energy cost on CIFAR-10. As shown in Figure 6, when an event triggers the inference, an exit will be selected according to the available energy and the energy cost of each exit. In this example, when Event 1 occurs, the stored energy is sufficient to support the inference to Exit 3, which is selected as the exit. However, when Event 2 occurs, the energy can only support the inference to Exit 1. At each exit, the confidence of the result is measured by entropy. If the confidence is higher than a threshold, the inference exits from this point. Otherwise, when more energy is available, an incremental inference will be made to proceed to the following exit for higher accuracy. In this example, since the confidence of Event 2 in Exit 1 is below the threshold, an incremental inference is conducted to proceed to Exit 2. This process alleviates the indefinitely long waiting time problem and an inference result with confidence can be obtained during each power cycle.

2.4.1.2 Optimization Goal

We use local inference to filter sensor readings from events so that only the interesting events are used to wake up the main device. Our figure of merit is the number of interesting events that are correctly processed in a fixed amount of harvested energy. We denote it as *IEpmJ*, or the number of Interesting Events per milliJoule. Maximizing *IEpmJ* is equivalent

to maximizing the average accuracy of all events:

$$IEpmJ = \frac{N_{correct}}{E_{total}} = \frac{\sum_{j=1}^{N_1} Correct_j + \sum_{j=1}^{N_2} 0}{E_{total}} = \frac{N}{E_{total}}(\frac{1}{N}\sum_{j=1}^{N} Correct_j), \tag{2-14}$$

where $N_{correct}$ is the number of correctly processed events. $N = N_1 + N_2$ is the number of all the events in which N_1 events are processed by inference and N_2 events are missed due to insufficient energy. $N_{correct}$ is a subset of N_1 and $N_{correct} = \sum_{j=1}^{N_1} Correct_j$. $Correct_j \in \{0, 1\}$ where $Correct_j = 1$ represents event j is correctly processed and $Correct_j = 0$ otherwise. Since N and E_{total} are constants determined by the EH environment, maximizing $IEpmJ$ is equivalent to maximizing the average accuracy of all N events, which is the number of correctly processed events over the total number of events $\frac{1}{N}\sum_{j=1}^{N} Correct_j$.

2.4.2 Model Compression for Energy Harvesting

To deploy the multi-exit networks to EH-powered devices, the model needs to be compressed to fit onto resource-constrained MCUs. Next, we will introduce how to apply the compression approach proposed in Section 2.3 to EH-powered devices.

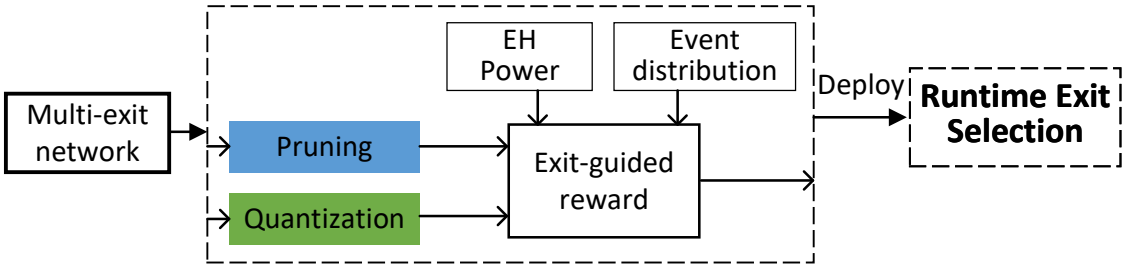


Figure 7: Compression with EH constraints and runtime exit selection after deployment.

Based on the proposed compression framework in Section 2.3, we will add EH constraints to develop an EH-powered trace-aware compression solution for EH applications. As shown in Figure 7, this approach takes the multi-exit network, EH power trace, and event distribution as the input and generates a non-uniform pruning rate and the bitwidth allocation policy for each layer. After compression, the network is deployed onto edge devices and the runtime algorithm will select the exit for each event, which will be introduced in Section 2.4.3.

Given a full-precision network with multiple early exits, we will explore the accuracy and energy cost allocation for each exit to maximize the average accuracy (equivalent to maximizing *IEpmJ* defined in Section 2.4.1) under the given EH power trace and event distribution. Some exits will be chosen more often than others. Thus, we will prioritize the accuracy of these exits during the compression process.

More specifically, the objective is to maximize the average accuracy of the given N events, which is the same as the goal Eq.(2-3) of the general framework. In addition to the original constraints Eq.(2-4) and Eq.(2-5), considering the typical power trace and event distribution, the additional energy constraints are as follows.

$$\text{s.t.} \quad E_i = f_E(\alpha_1, b_1^w, b_1^a, ..., \alpha_{L_i}, b_{L_i}^w, b_{L_i}^a), \quad \forall i \in \{1...m\}, \tag{2-15}$$

$$\sum_{j=1}^{n} EH_j \geq \sum_{j=1}^{n} E_{exit(j)}, \forall n \in \{1...N\}. \tag{2-16}$$

The constraint listed in Eq.(2-15) is that the energy cost E_i of exiting from exit i is determined by all the pruning rates and bitwidth allocations before this exit. The constraint in Eq.(2-16) means that for each of the N events, the total harvested energy from the beginning to the current time is greater than or equal to the total energy cost for all the happened events. Here, EH_j is the harvested energy after event $j-1$ and before event j. $E_{exit(j)}$ is the energy cost when exiting from exit i following policy $i = exit(j)$ for data sample j.

The reward function defined in Eq.(2-9) and Eq.(2-10) does not need to be changed, but R_{acc} defined in Eq.(2-8) has a different meaning. The percentage p_i of exit i being selected is determined by both the power trace EH_j and event sequence $\{1...N\}$ in Eq.(2-16). R_{acc} not only means to maximize the average accuracy of all events under the given power trace and event distribution, but also means to maximize *IEpmJ* defined in Eq.(2-14).

2.4.3 Runtime Exit Selection and Incremental Inference with Harvested Energy

During the compression process, the exit selection for an event j is determined statically using a static policy, e.g. a lookup table (LUT). However, naively following the static policy during runtime can result in low average accuracy in the long term. For example, when the EH power is low in the long run, even if the system has sufficient energy to select the exit

with the highest accuracy and energy cost for the current inference, a better decision can be selecting an exit with a lower energy cost to reserve energy for following events. This dynamic exit selection can improve the average accuracy. Besides, if the confidence at the selected exit is low, an incremental inference by proceeding to the following exit can improve the accuracy. We propose an online algorithm to make these two sequential decisions.

During runtime, both the power trace and the event distribution are unknown in advance. To select the best exit for each event, we propose to employ a lightweight RL algorithm, Q-learning [106]. Q-learning consists of the state set $\mathcal{S}$, the action set $\mathcal{A}$, and the reward function R. The state set $\mathcal{S}$ contains the currently available energy E and the charging efficiency P. Since both E and P are continuous values, to make the number of elements in $\mathcal{S}$ finite, we discretize E and P with appropriate step sizes. The action set $\mathcal{A}$ represents all the possible exits, which is $\mathcal{A} = \{exit_1, ..., exit_m\}$. The reward R is the accuracy of the selected exit $r = Acc_a, a \in \mathcal{A}$. The agent aims to learn the optimal policy π such that $a = \pi(s), a \in \mathcal{A}, s \in \mathcal{S}$ to maximize the reward $R = \sum r$. When an event happens, the agent takes two steps, one for selecting the action and the other for updating the Q-table. The action for the exit is selected by finding the highest Q-value in the current state, represented as $a = \arg\max_{a \in \mathcal{A}} Q(s, a)$, where $Q(s, a)$ denotes the Q-value of action-state pair (s, a). The entry (s, a) in the Q-table is updated as:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a \in \mathcal{A}} Q(s', a) - Q(s, a)). \tag{2-17}$$

The overhead of Q-learning is negligible. It only needs a lookup table (LUT) with state-action pairs as the entries, and the learning process is updating the LUT by Eq.(2-17).

To further improve the average accuracy, a second decision is made at the chosen exit for event j. If the confidence of the result is low and the remaining energy is high, the algorithm can decide to propagate the input further to the next exit for higher accuracy. The decision is made based on the confidence of the result and currently available energy. We use the entropy of the result as the measure of confidence [99]. We use another Q-table to make the decision.

2.5 Experiments

We conduct extensive experiments in the EH application scenario to demonstrate the effectiveness of our approaches in terms of *nonuniform compression*, *IEpmJ and accuracy*, *FLOPs and latency*, and *runtime adaptation*.

2.5.1 Experimental Setup

The experiments are targeting TI MSP432 MCU. To power the MCU, we use a solar profile from [71]. The backbone of the multi-exit model is LeNet [60]. We use LeNet because most SOTA DNNs designed for mobile devices cannot fit into typical MCUs even after compression. For example, MobileNetV2 [89] and DARTS [67] require 4.6MB and 6.6MB weight storage, respectively. However, a typical MCU has tens of KBs of weight storage. We extend LeNet to four convolutional layers and equip it with two early exits along the data path. The original network needs 580KB weight storage when represented with 32-bit floating-point numbers. The FLOPs of the three exits are 0.4452M, 1.2602M, and 1.6202M with corresponding accuracy of 64.9%, 72.0%, and 73.0%. The energy cost is 1.5mJ per million FLOPs. We are using the CIFAR-10 dataset and 500 events are randomly distributed across the duration of the EH power trace.

2.5.2 Nonuniform Pruning and Quantization

Our approach effectively finds out the pruning rate and quantization bitwidth allocation policy to maximize the average accuracy under the model size and FLOPs constraint. Figure 8 shows the layer-wise preserve rate and quantization bitwidth. The FLOPs constraint is set to 1.15M FLOPs, and the target model size is set to 16 KB. Under these constraints, our approach efficiently allocates the limited FLOPs and weight size budge to maximize accuracy. For pruning, the convolutional layers are pruned more because they are more FLOPs-intensive than the fully-connected layers. Different from pruning, quantization allocates more accuracy to convolutional layers by setting their bitwidth to 8. FC-B21 and FC-B31 are quantized to 1-bit possibly because they have large weight sizes and are less sensitive to data precision.

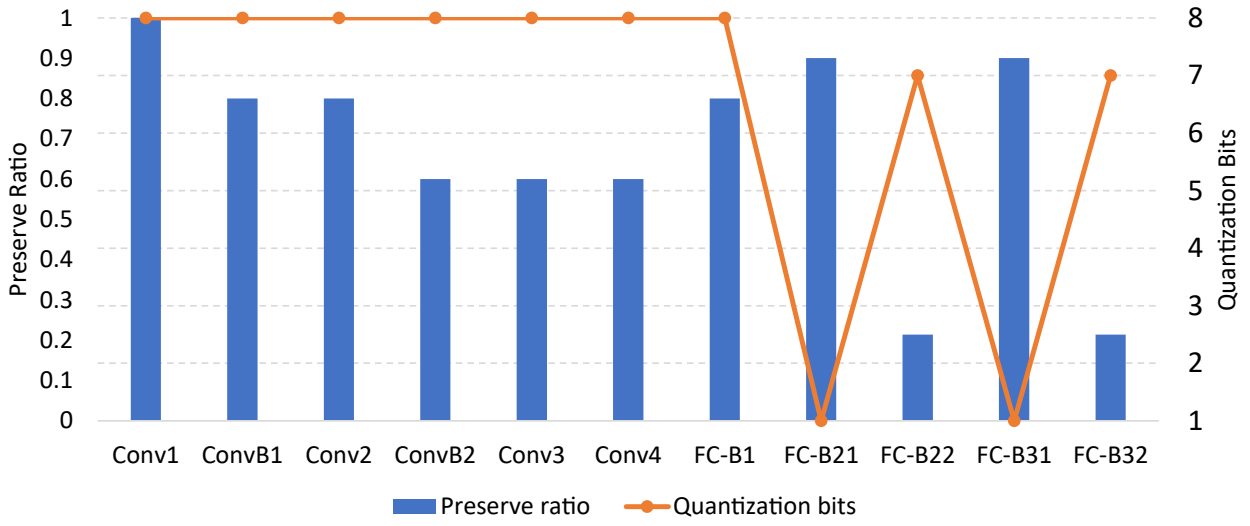


Figure 8: Pruning and quantization policy under 1.15M FLOPs and 16KB weight size constraints.

The search takes 6 hours on an Nvidia P100 GPU.

2.5.3 IEpmJ and Average Accuracy

The proposed approaches substantially outperform the SOTA baselines in terms of *IEpmJ* (Interesting Events per milliJoule) and equivalently the average accuracy of all events. We compare with three baselines. SonicNet is from the SOTA intermittent inference framework [27]. SpArSeNet is a network generated by a Neural Architecture Search framework for MCUs [24]. LeNet-Cifar is the LeNet [60] adapted for the CIFAR-10 dataset.

The result of *IEpmJ* is shown in Figure 9. Our approach outperforms SonicNet, SpArSeNet, and LeNet-Cifar by 3.6x, 18.9x, and 0.28x, respectively. Our approach achieves 0.89 interesting events per millijoules, while SonicNet and SpArSeNet only achieve 0.25 and 0.05, respectively. During compression, our approach considers the accuracy and energy cost of each exit, the EH power trace, and the event distribution to compress the network such that the *IEpmJ* is maximized. In terms of the accuracy of all events, where the accuracy of the missed event is

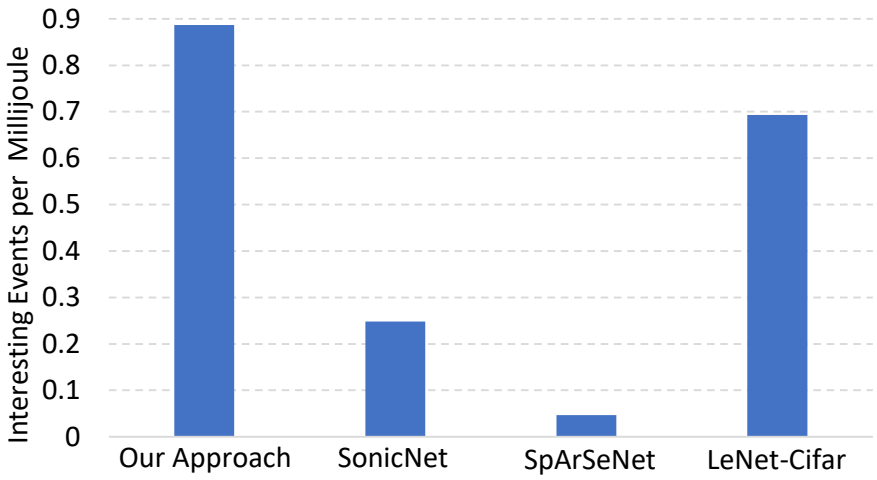


Figure 9: The number of interesting events per energy harvesting millijoule.

set to 0, our approach achieves an average accuracy of 50.1%, while SonicNet, SpArSeNet, and LeNet-Cifar only achieve 14.0%, 2.6%, and 39.2%, respectively. As for the accuracy of all the processed events, our approach achieves 65.4%, slightly lower than 75.4%, 82.7%, and 74.7% by the baselines. This is because we aim to improve the long-term accuracy to maximize *IEpmJ* instead of the accuracy for a single event. Solely aiming at the per-inference accuracy will generate a network with high energy cost and result in a high percentage of missed events, which degrades *IEpmJ*.

2.5.4 FLOPs and Latency

2.5.4.1 FLOPs

Our approach effectively reduces the FLOPs of each exit to maximize the average accuracy of all events. Reducing FLOPs is important because, with lower FLOPs and lower energy cost per inference, the saved energy can be allocated to other events which could have been missed due to insufficient energy. Figure 10 shows the FLOPs of each exit before and after compression. The FLOPs are reduced by 0.31x, 0.44x, and 0.67x for three exits, respectively. The reduction ratio of each exit is automatically decided by our approach. Different from our

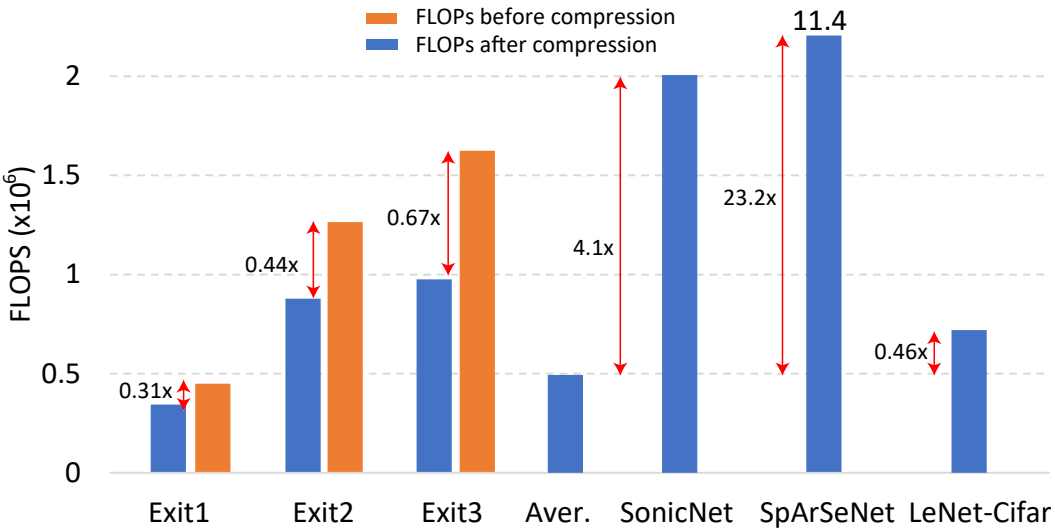


Figure 10: FLOPs reduced by compression.

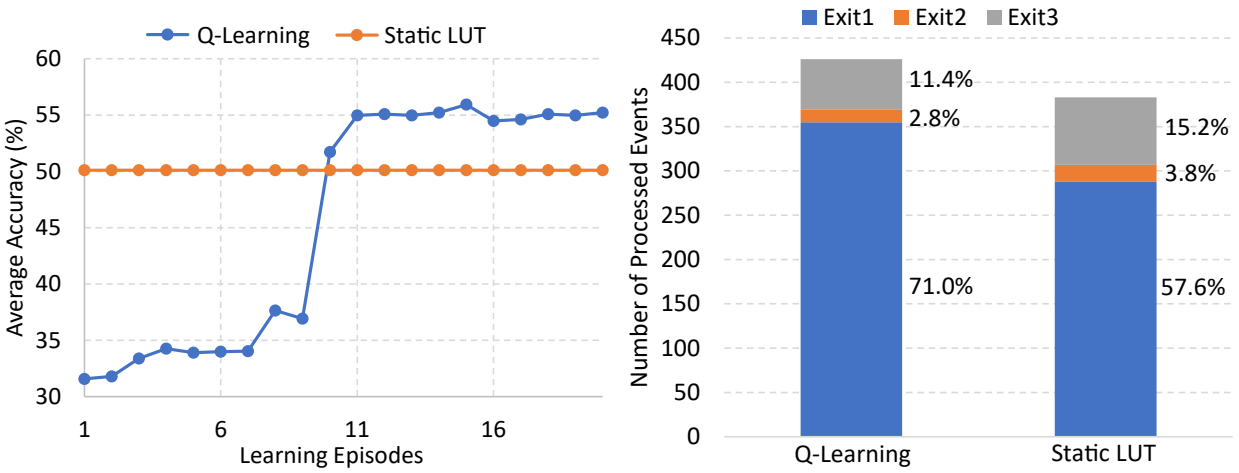
approach, the SonicNet has 2.0M FLOPs and SpArSeNet has 11.4M FLOPs because they did not consider the limited EH energy and only prioritize the per-inference accuracy. This results in high energy cost per inference, low *IEpmJ*, and low average accuracy across all events because a large portion of the events is missed. The LeNet-Cifar is manually designed by domain experts and has low FLOPs, which fortunately fits the EH scenario well.

2.5.4.2 Latency

Our approach greatly reduces both *per-event* latency and *per-inference* latency. First, the *per-event* latency is from the occurrence of an event to the end of inference. Across all the processed events, our approach improves the per-event latency by 7.8x, 10.2x, and 3.15x over three baselines. More specifically, the average latency of our approach is 18.0 time units (1 second per time unit), while the latency of the three baselines is 139.9, 183.4, and 56.7 time units, respectively. The improvement shows our approach smartly selects the early exits to quickly output a result when the EH energy is low, instead of waiting for multiple power cycles to reach the final exit as the baselines do. Second, our approach also improves the *per-inference* latency, which is from the start to the end of an inference. As shown in

Figure 10, using the FLOPs as the proxy for the *per-inference* latency, our approach improves the average *per-inference* latency by 4.1x, 23.2x, and 0.46x over three baselines.

2.5.5 Runtime Adaptation



(a) Runtime learning process.

(b) Number of processed events.

Figure 11: Runtime adaptation by lightweight learning.

The average accuracy of all events is further improved by the runtime exit selection. The runtime adaptation effectively learns from the EH environment and selects the exit for each event to maximize the average accuracy. The adaptation approach outperforms the static LUT by 10.2%. Figure 11a shows the average accuracy of all events is improved during the runtime adaptation. The lightweight Q-learning approach gradually learns to optimize the exit selection. Figure 11b shows the percentage and number of events exiting from each of the three exits. Compared with the static LUT, the Q-learning approach prioritizes exit 1 shown in the blue bar to decrease the energy cost of each inference. By strategy adaptation, the Q-learning approach processes 11.2% more events than the static LUT. The overhead of Q-learning is negligible by updating its Q-table.

2.6 Video Demo

To demonstrate the effectiveness of the proposed model compression framework for multi-exit neural networks, we have created video demos [4,5]. Our model compression framework is applicable to the general tiny edge devices, but we have found that it works particularly well for energy harvesting scenarios. Therefore, we demonstrate our methods using this scenario.

2.6.1 System Setup

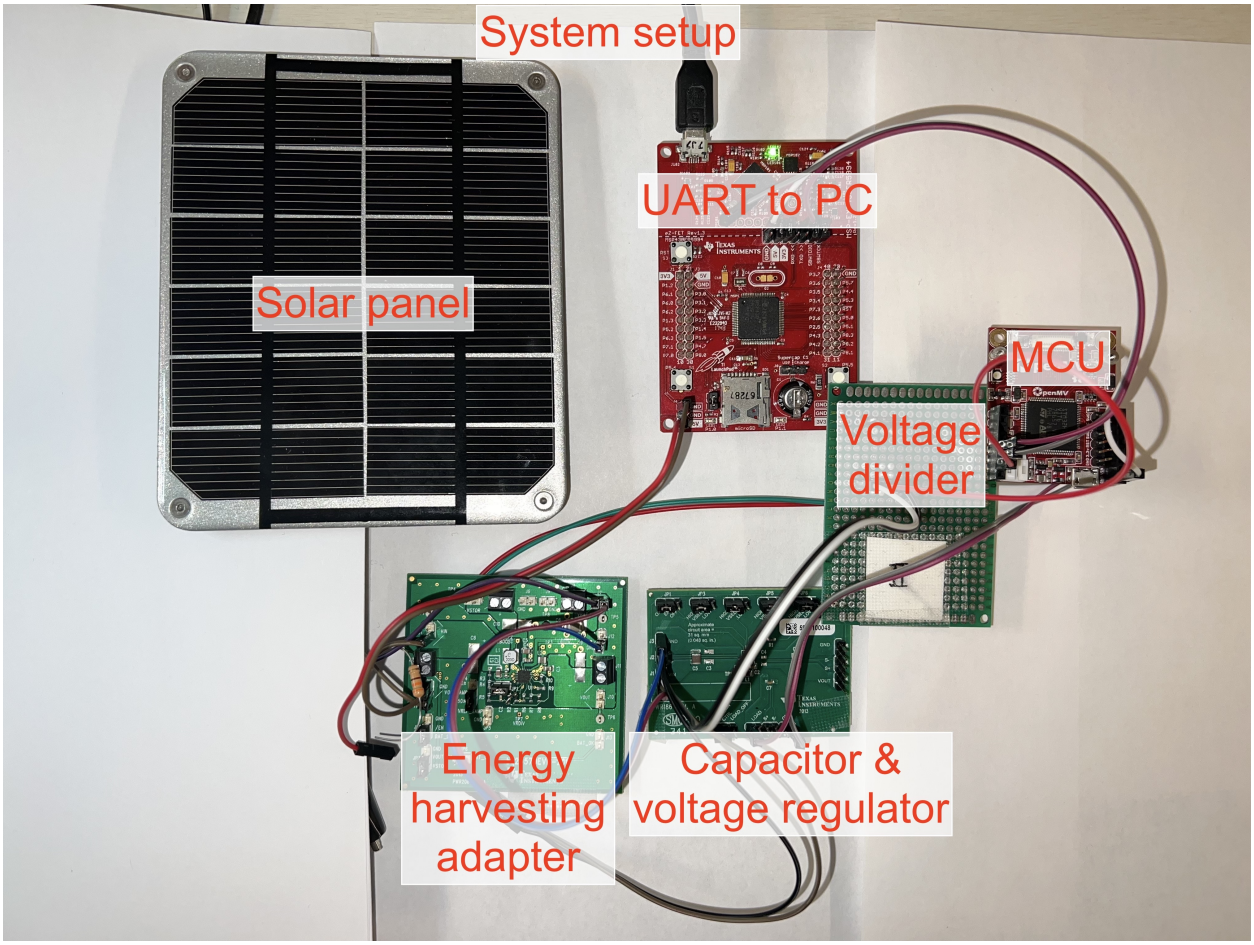


Figure 12: This is the system setup for the video demo of the model compression framework designed for multi-exit neural networks. The system is powered by energy harvesting technology.

The system setup is shown in Figure 12. The system is powered by a solar panel and its outputs are uploaded to a computer via the serial port. The system consists of an energy harvesting adapter, a capacitor, a voltage regulator, a voltage measurement module, and an MCU. The energy harvesting adapter, BQ25570 manufactured by Texas Instruments, is used to extract power from the solar panel to charge up the capacitor. The capacitor is a $1.5F$ supercapacitor, and the voltage regulator is TPS62740. The MCU running the neural network is STM32H7. The input images are pre-loaded onto the Flash memory of the MCU.

There are two video demos for this framework. The first video showcases how the proposed methods work for an easy input sample, and the second video demonstrates the effectiveness of our methods for a hard input sample. For these demos, we use the MNIST dataset [61] for this demo, and the multi-exit model is based on LeNet [59], with two early exits added.

2.6.2 An Easy Sample

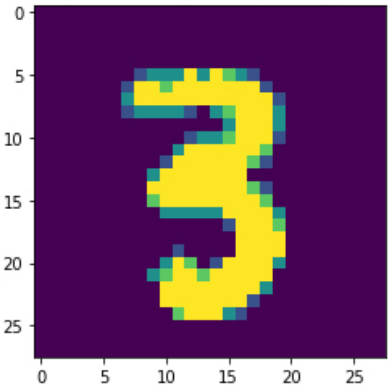


Figure 13: The easy sample used in the video demo.

We have created a video demo showcasing an easy sample, which is available online at [5]. The easy sample used in the video demo is shown in Figure 13. This digit is well-written and easily recognizable. For this easy sample, the model is able to obtain a confident and correct result at exit 1.

The event with this input image occurs three times. The system has enough energy to make a prediction in the first two instances. However, in the third instance, the system does not have enough energy and must wait for a while to accumulate enough energy.

2.6.3 A Hard Sample

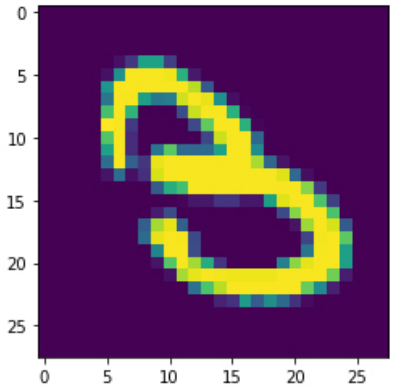


Figure 14: The hard sample used in the video demo.

We have created a video demo showcasing a hard sample, which is available online at [4]. The hard sample used in the video demo is shown in Figure 14. This digit is sloppily written and difficult to recognize. For this hard sample, the model's inference results from the first two exits have low confidence and are incorrect. The system, therefore, prefers to reach the last exit to obtain a confident and correct result.

This event occurs three times. The first time, the system has enough energy to make a prediction at all three exits. The system measures the confidence of prediction at exit 1 and decides to proceed to exits 2 and 3. The second time, the system does not have enough energy to make a prediction at all three exits. At exit 1, the system is not confident and must wait for a while to accumulate energy. Unfortunately, no more energy is available, so the system exits at exit 1 and informs the user about its prediction and the low confidence level.

In the third instance, the system does not have enough energy to make a prediction at all three exits. However, the system is able to accumulate enough energy to proceed to exit 2 and then exit 3. This allows the system to obtain a confident and correct prediction. At exit 1, the system was not confident and had to wait for a while to accumulate more energy. Fortunately, more energy became available, which allowed the system to proceed to exits 2 and 3.

2.7 Summary

This project aims to improve the efficiency of on-device inference by compression of multi-exit neural networks. We provide a model compression method for multi-exit neural networks with nonuniform compression ratios for each layer. Based on the proposed model compression techniques, a case study of applying the proposed model compression techniques to energy harvesting powered devices to enable event-driven IoT systems is introduced. The experimental results show superior accuracy and latency compared with SOTA techniques.

3.0 Efficient Supervised On-device Training

This chapter presents a project that enables the training of CNNs on edge devices by reducing the computation cost at training time [109]. It is organized as follows. First, the motivation is presented. Next, the background and the related works are introduced. Then, the details of the proposed data selection and the pruning of training computations to reduce the training cost are presented. After that, the experimental results are shown to demonstrate the effectiveness of the proposed techniques for reducing the training cost and preserving accuracy. Finally, the conclusion is presented to summarize this project.

3.1 Introduction

The maturation of deep learning has enabled on-device intelligence for IoT devices. CNN, as an effective deep learning model, has been intensively deployed on edge devices to extract information from sensed data, such as smart cities [96], smart agriculture [126], and wearable and medical devices [12,103]. The models are initially trained on high-performance computers (HPCs) and then deployed to IoT devices for inference. However, in the physical world, the statically trained model cannot adapt to the real world dynamically and may result in low accuracy for new input instances. On-device training has the potential to learn from the environment and update the model in situ. This enables incremental/lifelong learning [85] to train an existing model to update its knowledge, and device personalization [86] by learning features from the specific user and improving model accuracy. Federated learning [74] is another application scenario of on-device training, where a large number of devices (typically mobile phones) collaboratively learn a shared model while keeping the training data on personal devices to protect privacy. Since each device still computes the full model update by an expensive training process, the computation cost of training needs to be greatly reduced to make federated learning realistic.

While the efficiency of training in HPCs can always be improved by allocating more

computing resources, such as 1024 GPUs [7], training on resource-constrained IoT devices remains prohibitive. The main problem is the large gap between the high computation and energy demand of training and the limited computing resource and battery on IoT devices. For example, training ResNet-110 [35] on a 32x32 input image takes 780M FLOPs, which is prohibitive for IoT devices. Besides, since computation directly translates into energy consumption and IoT devices are usually battery-constrained [30], the high computation demand of training will quickly drain the battery. While existing works [45, 80, 108] effectively reduce the computation cost of inference by assigning input instances to different classifiers according to the difficulty, the computation cost of training is not reduced.

To address this challenge, this work aims to enable on-device training by significantly reducing the computation cost of training while preserving the desired accuracy. Meanwhile, the proposed techniques can also be adopted to improve training efficiency on HPCs. To achieve this goal, we investigate the computation cost of the entire training cycle, aiming to eliminate unnecessary computations while keeping full accuracy. We made the following two observations: *First*, not all the input instances are important for improving the model accuracy. Some instances are similar to the ones that the model has already been trained with and can be completely dropped to save computation. Therefore, developing an approach to filter out unimportant instances can greatly reduce the computation cost. *Second*, for the important instances, not all the computation in the training cycle is necessary. Eliminating insignificant computations will have a marginal influence on accuracy. In the backward pass of training, some channels in the error maps have small values. Pruning out these insignificant channels and corresponding computation will have a marginal influence on the final accuracy while saving a large portion of computation.

Based on the two observations, we propose a novel framework consisting of two complementary approaches to reduce the computation cost of training while preserving full accuracy. The first approach is an early instance filter to select important instances from the input stream to train the network and drop trivial ones. The second approach is error map pruning to prune out insignificant computations in the backward pass when training with the selected instances.

In summary, the main contributions of this paper include:

- **A framework to enable on-device training.** We propose a framework consisting of two approaches to eliminate unnecessary computation in training CNNs while preserving full network accuracy. The first approach improves the training efficiency of both the forward and backward passes, and the second approach further reduces the computation cost in the backward pass.
- **Self-supervised early instance filtering (EIF) on the data level.** We propose an instance filter to predict the loss of each instance and develop a self-supervised algorithm to train the filter. Instances with predicted low loss are dropped before starting the training cycle to save computation. To train the filter simultaneously with the main network, we propose a self-supervised training algorithm including the adaptive threshold-based labeling strategy, uncertainty sampling-based instance selection algorithm, and weighted loss for biased high-loss ratio.
- **Error map pruning (EMP) on the algorithm level.** We propose an algorithm to prune insignificant channels in error maps to reduce the computation cost in the backward pass. The channel selection strategy considers the importance of each channel on both the error propagation and the computation of the weight gradients to minimize the influence of pruning on the final accuracy.

We evaluate the proposed approaches on networks of different scales. ResNet and VGG are for on-device training of mobile devices, and LeNet is for tiny sensor node-level devices. The experimental results demonstrate that the proposed approaches effectively reduce the computation and energy costs of training with little or no impact on model accuracy.

3.2 Background and Related Work

3.2.1 Background of CNN Training

The training of CNNs is most commonly conducted with the mini-batch stochastic gradient descent (SGD) algorithm. It updates the model weights iteration-by-iteration using a mini-batch (e.g. 128) of input instances. For each instance in the mini-batch, a forward

pass and a backward pass are conducted. The forward pass attempts to predict the correct outputs using current model weights. Then the backward pass back-propagates the loss through layers, which generates the error maps for each layer. Using the error maps, the gradient of the loss w.r.t. the model weights are computed. Finally, the model weights are updated by using the weight gradients and an optimization algorithm such as SGD.

To provide labeled data for on-device training, labeling strategies from existing works can be used. For example, the labels can come from aggregating inference results from neighbor devices [62] (e.g. voting), employing spatial context information as the supervisory signals [77], or naturally inferred from user interaction [31, 72] such as next-word-prediction in keyboard typing.

3.2.2 Related Work

3.2.2.1 Accelerated Training

There are a number of works on accelerating network training. Stochastic depth [41] accelerates the training by randomly bypassing layers with the residual connection. E2Train [105] randomly drops mini-batches and selectively skips layers by using residual connections to save computation costs. Different from [105], which randomly drops mini-batches, we investigate the importance of each instance before keeping or dropping it. The input data from the real world is not ideally shuffled and valuable instances for training can concentrate within one mini-batch. Simply dropping mini-batches can miss important instances for training the network. Besides, the layer skipping in these two works relies on the ResNet architecture [35], and cannot be naturally extended to general CNNs. In contrast, our approaches are applicable to general CNNs. OHEM [94] selects high-loss instances and drops low-loss ones to improve training efficiency. It computes the loss values of all instances in the forward pass and only keeps high-loss instances for the backward pass. The main drawback is that the computation in the forward pass of low-loss instances is wasted. Different from this, our approach predicts the loss of each instance and drops low-loss instances before starting the forward pass, which eliminates the computation cost of low-loss instances. ECP [93] accelerates training by selective convolution. However, this approach does not target on-device training and is only

evaluated on a small-scale dataset.

3.2.2.2 Distributed Training

Another way to accelerate training is by leveraging distributed training [104] with abundant computing resources and large batch sizes. [7] employs an extremely large batch size of 32K with 1024 GPUs to train ResNet-50 in 20 minutes. [48] integrates a mixed-precision method into distributed training and pushes the time to 6.6 minutes. However, these works target leveraging highly-parallel computing resources to reduce the training time and actually increase the total computation cost, which is infeasible for training on resource-constrained IoT devices.

3.2.2.3 Network Pruning during Training

Some works aim to train and prune the network architecture simultaneously. [9] and [70] aim to accelerate training by reconfiguring the network to a smaller one during training. The main drawback is that the network is pruned on the offline training dataset, and the ability of the pruned network for further on-device learning is compromised. Instead, we focus on reducing the computation cost of online training and the entire network architecture is preserved to keep the full ability for learning in an uncertain future.

3.2.2.4 Neural Architecture Search

There are extensive explorations on neural architecture search (NAS). [50, 51, 107, 119] search neural architectures for hardware-friendly inference. [68] further considers quantization during NAS for efficient inference. However, these works only aim to design network architectures for efficient inference. The computation cost of training is not considered.

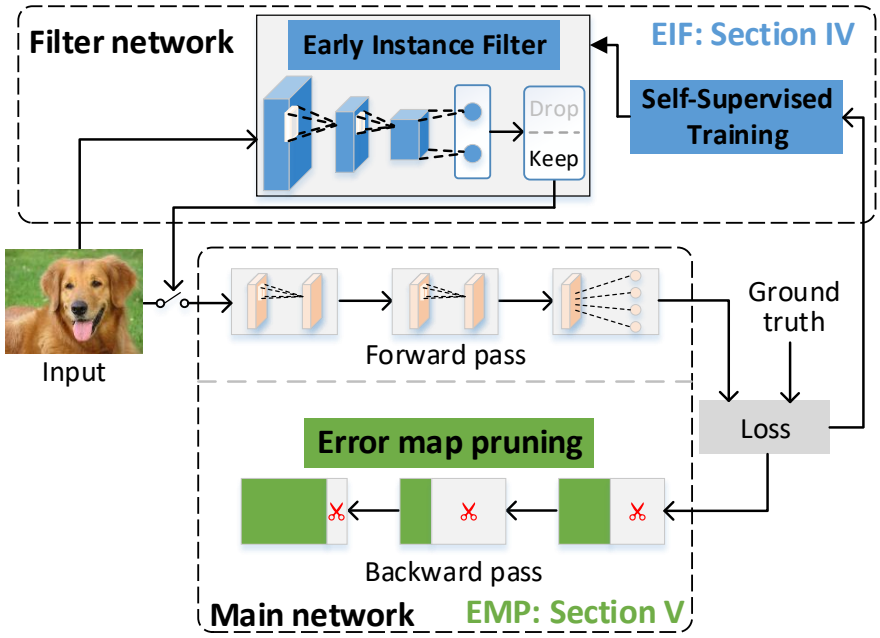


Figure 15: Overview of early instance filtering (EIF) and error map pruning (EMP).

3.3 Framework Overview

The overview of the proposed framework is shown in Figure 15. On top of the main neural network, a small instance filter network is proposed to select important instances from the input stream to train the network and drop trivial ones. When the input instances arrive, the early instance filter predicts the loss value for each instance as if the instance was fed into the main network and makes a binary decision to drop or preserve this instance. If the predicted loss is high and the instance is preserved, the main network will be invoked to start the forward and backward pass for training. Since the loss prediction is for the main network, once the main network is updated, the instance filter also needs to be trained for accurate loss prediction. The training of the instance filter is self-supervised based on the labeling strategy by the adaptive loss threshold, instance selection by uncertainty sampling, and the weighted loss for biased high-loss ratio, which will be introduced in Section 3.4. Once important instances are selected, the error map pruning further reduces the computation cost

of the backward pass. It prunes out channels in the error maps that have small contributions to the error propagation and gradient computation, which will be introduced in Section 3.5.

3.4 Self-Supervised Early Instance Filter

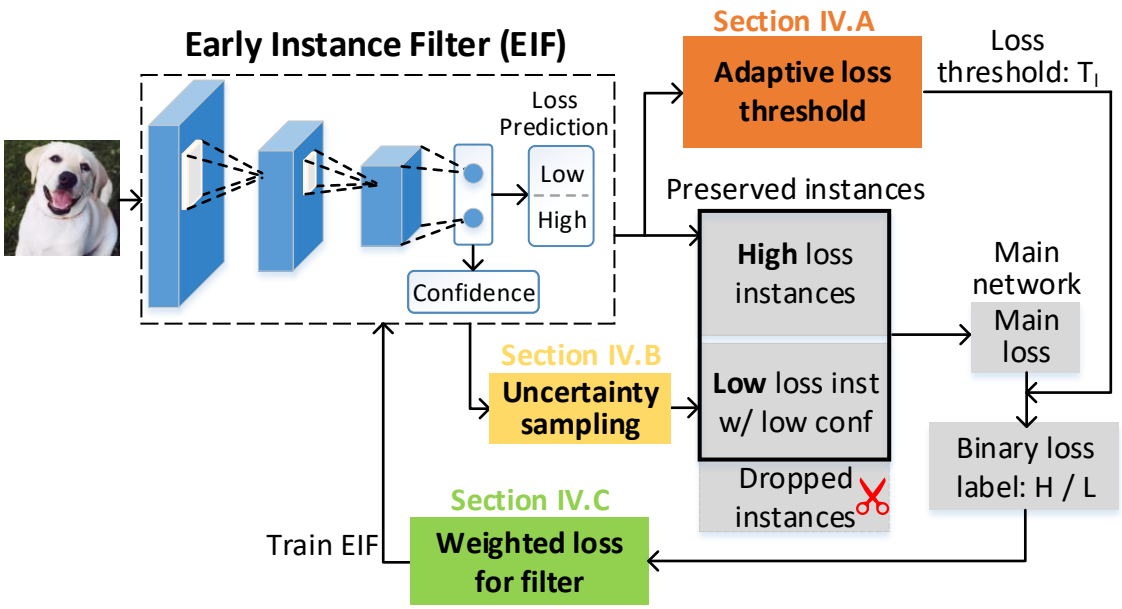


Figure 16: Self-supervised training of early instance filter (EIF) by adaptive loss threshold, uncertainty sampling, and weighted loss.

The early instance filter (EIF) is used to select important instances for training the main network and drop trivial instances to reduce the computation cost of training. Since the main network is constantly being updated during training, it is essential to tune the EIF every time the main network is updated. In this way, the EIF can accurately select important instances based on the latest state of the main network. In this section, we will first introduce the working flow of EIF to select instances for training the main model. Then we discuss the challenges of updating the EIF. After that, we present three approaches to address these challenges such that the EIF can be effectively updated.

To select important instances and drop trivial ones on-the-fly during training, the EIF

predicts the loss value of each instance from the input stream without actually feeding the instance into the main network. Trivial instances with predicted low loss are dropped before the forward pass, which eliminates the computation on the forward pass and more computationally intensive backward pass of the main network. Important instances with predicted high loss are preserved to complete the forward pass, calculate the loss, and finish the backward pass to compute the weight gradients to update the main network. Kindly note that the instances are not pre-selected before the training starts. Instead, they are selected on-the-fly during training based on what the main network has and has not learned at the current state.

Figure 16 shows the working flow of EIF. The user first needs to pre-define a high-loss ratio R_{set} (e.g. 10%) such that these amounts of instances in the whole input stream will be predicted as high-loss and the others will be predicted as low-loss. Only instances predicted as high-loss will be used for training the main network. When instances arrive sequentially, the early instance filter predicts the loss value of each instance i as binary high or low $y_{pred,i} = \{H, L\}$ for the main network such that the pre-defined high-loss ratio is satisfied. The filter also produces the confidence of each loss prediction, represented by the entropy of the loss prediction. Since the loss prediction by the EIF network is for the main network and the main network is constantly being updated, it is essential to re-train the EIF network every time the main network is updated to realize accurate loss prediction. However, there are several challenges in realizing automatic self-supervised training for the EIF network. In this section, we will first present three major challenges. Then, we will present three techniques to address these challenges: adaptive loss threshold, uncertainty sampling, and weighted loss, as shown in Figure 16.

3.4.1 Challenges

During on-device training, instances with predicted low loss are dropped before feeding to the main network to compute the actual loss, and their true loss values are unknown. Thus, we can only know the true loss values of instances with predicted high loss, which brings two challenges. The *first* challenge is how to label instances as high-loss or low-loss for training

the EIF according to the pre-defined high-loss ratio. For example, if we could know the loss values of all instances, defining a loss threshold to separate 10% instances with the highest loss values is simply sorting all the loss values and finding the value for separation. Since the loss values of dropped instances are unknown, defining a loss threshold remains a challenge.

The *second* challenge is that the EIF network can choose what instances will be used to train itself, which is not possible for normal CNN training. As long as the EIF network is not 100% accurate, it will make wrong predictions. To avoid punishment, instead of adjusting its own weights to make accurate loss predictions, the filter will learn a shortcut by predicting all the new input instances as low loss and dropping them. Since the dropped instances will not be fed to the main network, the EIF network will never know the ground truth of the losses and thus it will not be punished for doing so. In this way, EIF will think it makes perfect predictions. Dropping all the new instances prevents further training of the filter and main network.

The *third* challenge is how to correctly train the filter when the number of high-loss and low-loss instances is extremely unbalanced in the input stream. This is different from normal training datasets such as CIFAR-10 and ImageNet, in which the number of instances in each class is balanced. The unbalanced number of high-loss and low-loss instances makes the EIF network training ineffective. For example, when the pre-defined high-loss ratio is relatively low (e.g. 10%), simply predicting all the instances as low-loss will produce high accuracy of 90% on the filter, which it believes is a good result. However, this prediction is useless since it does not find any important instance to train the main network.

We will present three techniques to address these challenges.

3.4.2 Adaptive Loss Threshold Based Labeling Strategy

The adaptive loss threshold is used to provide the ground truth (labels) for training the EIF. With the adaptive loss threshold, we can label the loss of instances as high-loss or low-loss to train the EIF. During the training of EIF and the main network, R_{set} percent of instances will be predicted as high-loss by the EIF. The true loss values of instances predicted as high-loss can be obtained on the main network. However, we do not know the true loss

values of instances predicted as low-loss since they are dropped before feeding into the main network. With only partial loss values, defining an exact loss threshold (e.g. sorting all loss values and finding the threshold) is challenging. Therefore, we aim to approximate the threshold. To achieve this, we first define true high (TH) instances as the instances with predicted high loss by the filter and labeled as high-loss by the loss threshold. We will monitor the number of TH instances in the last n mini-batches. Then we calculate the percentage R_{TH} as the number of TH instances in the preserved ones over all the instances in the last n mini-batches. By comparing the percentage R_{TH} with the pre-defined percentage R_{set}, the loss threshold is adjusted to draw R_{TH} to the pre-defined percentage R_{set}.

Formally, with adaptive loss threshold T_l, instances are labeled as high-loss or low-loss as follows.

$$y_i = \begin{cases} H & \text{if } loss_i \geq T_l, \\ L & \text{otherwise,} \end{cases} \tag{3-1}$$

where $loss_i$ is the loss value of instance i computed by the main network. T_l is the adaptive loss threshold.

The true high (TH) loss instance ratio R_{TH} by the filter is defined as follows.

$$R_{TH} = \frac{1}{mn} \sum_{i=1}^{mn} \mathbb{I}(y_{pred,i} = H)\mathbb{I}(y_i = H), \tag{3-2}$$

where $\mathbb{I}(x)$ is an indicator function which equals 1 if x is true and 0 otherwise. $y_{pred,i}$ is the binary prediction by the filter for instance i, and y_i is the loss label by Eq.(3-1). m is the batch size, and n is the number of mini-batches to monitor for one update of the loss threshold.

Based on the computed R_{TH} and pre-defined R_{set}, the loss threshold T_l is adjusted to draw R_{TH} to R_{set}. When R_{TH} is larger than R_{set}, too many instances are labeled and predicted as high loss, which indicates T_l is too small. Therefore, T_l will be incremented by multiplying with a factor larger than 1. Similarly, when R_{TH} is smaller than R_{set}, T_l is too large and will be attenuated. The loss threshold T_l is adjusted as:

$$T_l = \begin{cases} \alpha_1 T_l & \text{if } R_{TH} \geq R_{set}, \\ \alpha_2 T_l & \text{otherwise,} \end{cases} \tag{3-3}$$

where α_1 and α_2 are two hyper-parameters where α_1 is larger than 1 and α_2 is smaller than 1 to define the step size.

The computed R_{TH} is essential to the self-supervised training of the EIF. More specifically, R_{TH} controls the loss threshold T_l by Eq.(3-3), which further controls the instance labels y_i by Eq.(3-1) for training the instance filter. With the labels y_i, the filter will be trained accordingly to predict high-loss instances. The number of instances with predicted high-loss by the filter and labeled as high-loss will be used to compute the new R_{TH} by Eq.(3-2), which further adjusts T_l. This process continues for each mini-batch, which forms the self-supervised training of the instance filter. Leveraging the self-supervision, the loss threshold T_l will be properly adjusted and the instance filter will be well-trained to track the latest state of the main network. In this way, the true high-loss ratio R_{TH} affected by both the filter and the loss threshold will be kept at the set ratio R_{set}. The filter will effectively select R_{set} percent important instances for training the main network.

3.4.3 Instance Selection by Uncertainty Sampling

The main reason for the second challenge is that if an instance is dropped, it will never be fed to the main network and the EIF network will never know the ground truth of the loss. In this way, the labels (i.e. high-loss or low-loss) of the dropped instances for training the EIF will be unknown, and the EIF cannot be correctly trained. To address this problem, we keep some instances with predicted low loss, which would be dropped, to augment the preserved instances for training the filter. In this way, wrong loss predictions of the dropped instances will also punish the filter, which forces it to actually learn to find important instances. To decide which instances to keep and minimize the number of selected instances, we employ uncertainty sampling [55]. The dropped instances that the filter is least confident about will be fed into the main network to compute the loss value. To measure the confidence of loss prediction by the filter, we use the entropy defined as:

$$entropy(i) = -\sum_{c\in\{H,L\}} p_{i,c}\log p_{i,c}, \quad p_{i,c} = prob(y_{pred,i} = c), \tag{3-4}$$

where $p_{i,c}$ is the computed probability by the filter of being high-loss ($c = H$) or low-loss ($c = L$) for instance i. The smaller the entropy, the more confident the filter is about the prediction.

Based on the entropy, we select from the dropped instances where the entropy is above the entropy threshold to augment the preserved instances for training the filter. The set of selected instances is defined as:

$$\mathcal{I} = \{i \mid i \in \{TL, FL\},\ entropy(i) > entropy_T\}, \tag{3-5}$$

where $entropy_T$ is the entropy threshold.

3.4.4 Weighed Loss for Biased High-Loss Ratio

To address the third challenge, we propose to use the weighted loss function to make the EIF training process fair in treating the high-loss instances when their ratio is low. In this way, the EIF can be trained to make accurate loss predictions and select important instances for training the main network.

Traditionally, for datasets with balanced classes, we use the average loss of each instance as the loss function of a mini-batch for training. In our case, based on the binary loss label y_i in Eq.(3-1) and the binary loss prediction $y_{pred,i}$ by the filter, the loss function for instance i is defined by cross-entropy as:

$$L_i = -\sum_{c \in \{H,L\}} \mathbb{I}(y_i = c) \log p_{i,c}, \tag{3-6}$$

where $p_{i,c}$ defined in Eq.(3-4) is the computed probability of being a high or low loss for instance i by the filter. L_i measures how well the loss prediction approximates the true loss label and will be minimized during training. The average loss will be the average loss value of each preserved instance in a mini-batch. However, when the pre-defined high-loss ratio is not 50% and makes the number of high-loss and low-loss instances unbalanced, directly using the average loss will result in effective training of the EIF.

To understand the inefficiency of training with the average loss, we define the weighted

loss for preserved instances in a mini-batch to train the filter as:

$$L = \sum_{i \in TH} w_H L_i + \sum_{j \in FH} w_L L_j + \sum_{p \in TL} w_L L_p + \sum_{q \in FL} w_H L_q, \tag{3-7}$$

where TH, FH, TL, and FL represent true high, false high, true low, and false low loss instances, respectively. TH and FH are instances with predicted high loss and are labeled as H and L by Eq.(3-1), respectively. TL and FL are instances with predicted low loss and selected by uncertainty sampling in Eq.(3-5), which have loss labels L and H, respectively. The weights w_H and w_L represent how important the true high loss (instances with loss label H, including TH and FL) and true low loss (instances with loss label L, including TL and FH) instances are, respectively. w_H and w_L are normalized such that the weights of all instances in Eq.(3-7) sum up to 1.

When the pre-defined high-loss ratio R_{set} is not 50%, the number of high-loss and low-loss instances will be not equal in the input stream. This makes training the EIF with the average loss ineffective. For example, when R_{set} is set to 10%, only 10% of the instances streamed in will be labeled as high-loss by the adaptive loss threshold. In this way, 90% of elements in Eq.(3-7) will be low-loss instances and dominate the loss. If we were using average loss, all the weights will be the same. To minimize the loss when training the filter, simply predicting all instances as low-loss will produce small loss values on the dominating second and third elements in Eq.(3-7), and hence the total loss, which prevents effective training of the filter.

To address this problem, we make the weights biased by setting $w_H = \frac{1}{R_{set}}$ and $w_L = \frac{1}{1-R_{set}}$. In this way, we have $w_H \times percent(H = TH + FL) = w_L \times percent(L = TL + FH)$. The first and fourth sums in Eq.(3-7) correspond to the high-loss (H) instances. The second and third sums in Eq.(3-7) correspond to the low-loss (L) instances. By setting the weights in this way, the high-loss and low-loss instances will contribute equally to the total loss and will be treated fairly in training. In the above example, while the first and fourth sums only contribute to 10% of the number of elements, the higher weight $w_H = \frac{1}{0.1} = 10$ makes them equally important as the second and third sums, which have lower weight $w_H = \frac{1}{0.9} = 1.1$. Therefore, the instance filter can be correctly trained with the unbalanced number of high-loss and low-loss instances and accurately predict high-loss ones.

With the predicted high-loss instances by the filter, the selected instances by uncertainty

sampling, and the weighted loss function for training, the filter is effectively trained to predict high-loss instances for training the main network.

3.5 Error Map Pruning in Backward Pass

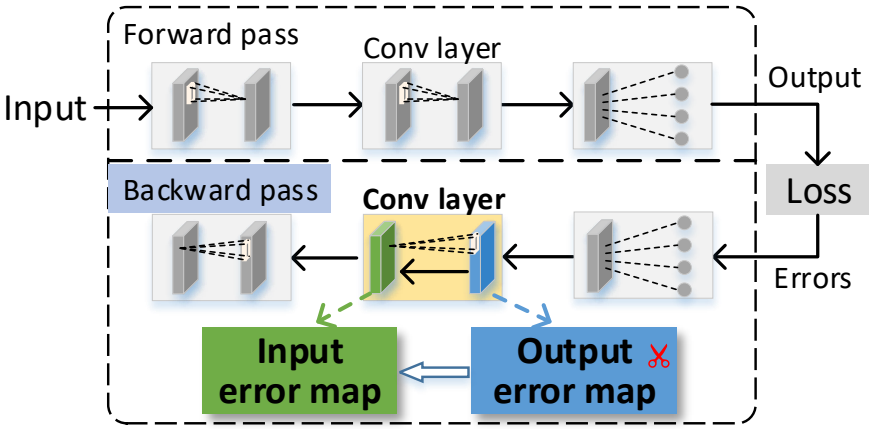


Figure 17: Error maps of convolutional layers in back-propagation.

When training with the selected instances, the computation in the backward pass can be further reduced by error map pruning (EMP). Since the backward pass takes about 2/3 of the computation cost of training, reducing its computation can effectively reduce the total cost. As shown in Figure 17, in the backward pass of training, the back-propagation propagates the errors layer-by-layer from the last layer to the first layer. We focus on pruning convolutional layers because they dominate the computation cost in the backward pass. Within one convolutional layer, the input error map is generated from the output error map of the same layer. The output error map consists of many channels. We aim to prune the insignificant channels to reduce the computation cost of training.

Given a pruning ratio, we need to keep the most representative channels in the error map to maintain as much information such that the training accuracy is retained. The proposed channel selection strategy aims to prune the channels that have the least influence on both error propagation and the computation of the weight gradients.

3.5.1 Channel Selection to Minimize Reconstruction Error in Error Propagation

3.5.1.1 Problem Formulation

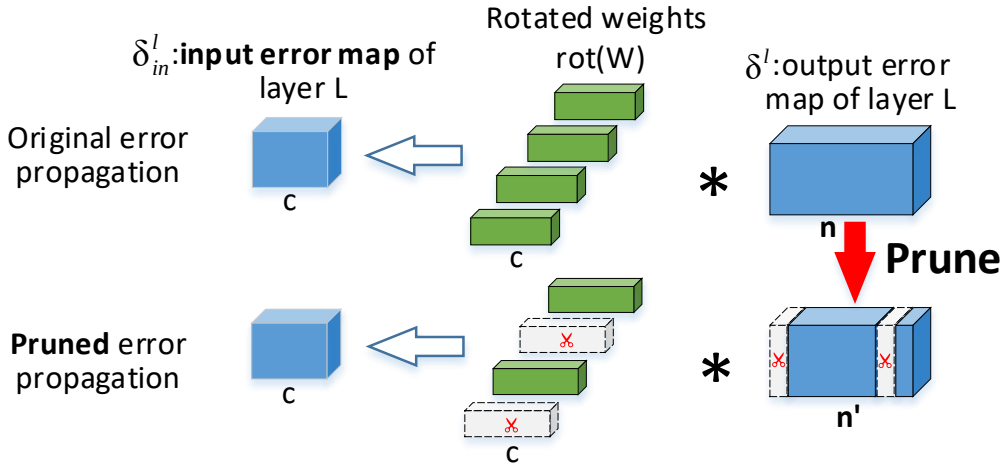


Figure 18: Back-propagation of errors with pruned error map.

The first criterion to select the channels to be pruned is to minimize reconstruction error in error propagation. The error propagation for one convolutional layer is shown at the top of Figure 18. Within one layer, the error propagation starts from the output error map δ^l shown on the right, convolves δ^l with the rotated kernel weights $rot(W^l)$, and generates the input error map δ^l_{in} on the left. The error propagation with pruned δ^l is shown at the bottom of Figure 18. The number of channels in δ^l is pruned from n to n'. When computing δ^l_{in}, the computations corresponding to the pruned channels, which are convolutional operations between δ^l and the rotated weights, are removed. To maintain training accuracy, it is desirable to keep the input error map δ^l_{in} as similar as possible before and after pruning. In other words, the reconstruction error on the input error map should be minimized.

Formally, without channel pruning of δ^l, δ^l_{in} is computed as follows.

$$\delta^l_{in} = \sum_{j=1}^{n} rot(W^l_j) * \delta^l_j, \tag{3-8}$$

where δ^l_{in} is the input error map consisting of c channels, each with shape $[W_{in}, H_{in}]$. $rot(W^l_j)$ is the rotated weights of jth convolutional kernel with shape $[c, k_w, k_h]$. δ^l_j is the jth channel

of the output error map with shape $[W, H]$.

Given a pruning ratio α and an output error map δ^l, we aim to reduce the number of channels in δ^l from n to n' such that $\alpha = n'/n$. To minimize the reconstruction error on δ^l_{in}, the channel selection problem is formulated as follows.

$$\arg\min_{\boldsymbol{\beta}} \left\| \delta^l_{in} - \sum_{j=1}^{n} rot(W^l_j) * (\delta^l_j \beta_j) \right\|_2, \tag{3-9}$$

$$\text{s.t.} \quad \|\boldsymbol{\beta}\|_0 = n', \tag{3-10}$$

where $\boldsymbol{\beta}$ is the error map selection strategy, represented as a binary vector of length n. β_j is the jth entry of $\boldsymbol{\beta}$, and $\beta_j = 0$ means the jth channel of δ^l_j is pruned. The ℓ_2-norm $\|\boldsymbol{x}\|_2 = \sqrt{\Sigma x^2}$ measures the reconstruction error on δ^l_{in}.

However, directly solving the minimization problem is prohibitive. δ^l_{in} in the problem is computed by Eq.(3-8), which completes all the computations in error propagation and defeats the purpose of saving computation. To select channels to prune before starting the actual error propagation, we define the importance score as an indication of how much each channel will influence the value of δ^l_{in} and prune the least important channels to minimize the reconstruction error on δ^l_{in}.

3.5.1.2 Importance Score

In Eq.(3-9), when a channel δ^l_j is pruned, the computation error on δ^l_{in} is caused by the pruned $rot(W^l_j) * \delta^l_j$. As a fast and accurate estimation of the magnitude of $rot(W^l_j) * \delta^l_j$, we define the importance score of channel j as follows.

$$s_j = \gamma_1 \left\| W^l_j \right\|_1 + \gamma_2 \left\| \delta^l_j \right\|_1, \tag{3-11}$$

where $\|W^l_j\|_1$ is ℓ_1-norm of convolutional kernel j, computed by $\sum_{i=1}^{c} |W^l_{j,i}|$. Here we remove the rotation on W^l_j since it does not change the ℓ_1-norm. $\|\delta^l_j\|_1$ is ℓ_1-norm of channel j in the output error map, computed by the sum of its absolute values $\sum_{x=1}^{W} \sum_{y=1}^{H} |\delta^l_{j,x,y}|$. γ_1 and γ_2 are two hyper-parameters to adjust the weight of each ℓ_1-norm.

The importance score s_j gives an expectation of the magnitude that a channel j in δ^l

contributes to δ^l_{in}. Channels with small magnitudes in δ^l and corresponding kernel weights $|W^l_j|$ tend to produce trivial values in the input error map δ^l_{in}, which can be pruned while minimizing the influence on δ^l_{in}.

3.5.2 Channel Selection to Minimize Reconstruction Error in Gradient Computation

3.5.2.1 Problem Formulation

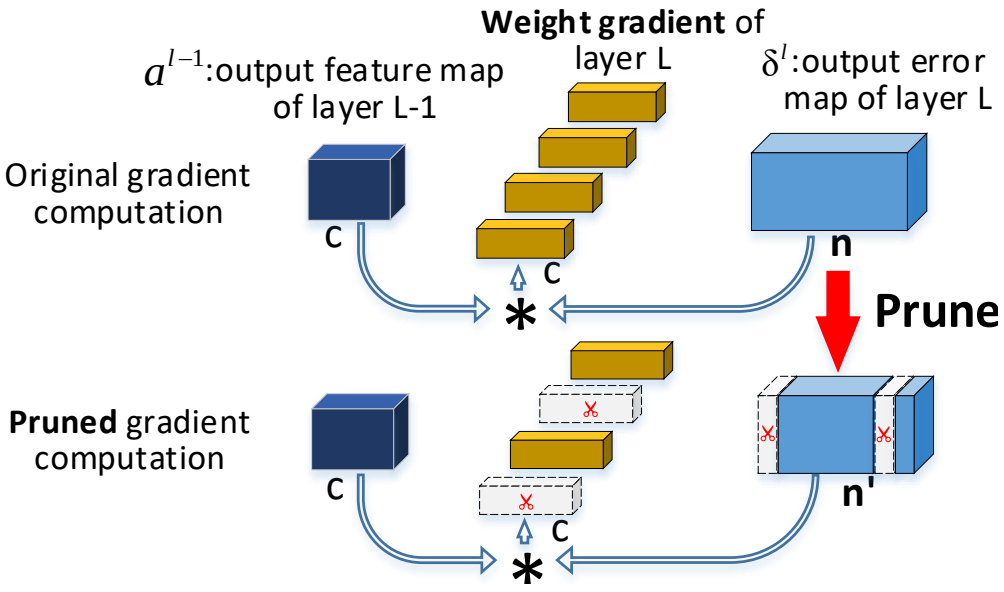


Figure 19: Computation of weight gradient with pruned error map.

The second criterion to select the channels to be pruned is to minimize the reconstruction error in the weight gradients. The computation of the weight gradients without pruning is shown at the top of Figure 19. The output feature map a^{l-1} of the previous layer convolves with one channel of the output error map δ^l to produce the gradient of one kernel. When some channels in δ^l are pruned, the computation of the weight gradients corresponding to the pruned channels is removed. To retain training accuracy, we want to keep the weight gradients before and after pruning as same as possible. Without channel pruning of δ^l, the

weight gradients of kernel j are computed as follows.

$$g_{w,j}^{l} = a^{l-1} * \delta_{j}^{l}, \quad \forall j \in \{1, ..., n\}, \tag{3-12}$$

where $g_{w,j}^{l}$ is the weight gradients of kernel j with shape $[c, k_w, k_h]$. a^{l-1} is the output feature map of the previous layer $l-1$ with shape $[c, W_{in}, H_{in}]$. δ_{j}^{l} is the channel j of the output error map in layer l, which has shape $[W, H]$.

To determine the channel selection strategy $\boldsymbol{\beta}$ while minimizing the reconstruction error on the gradient computation, the channel selection problem is formulated as:

$$\arg\min_{\boldsymbol{\beta}} \sum_{j=1}^{n} \left\| g_{w,j}^{l} - a^{l-1} * (\delta_{j}^{l} \beta_j) \right\|_2, \quad \text{s.t. } \|\boldsymbol{\beta}\|_0 = n'. \tag{3-13}$$

Similar to Eq.(3-9), we use the ℓ_2-norm $\|\cdot\|_2$ to measure the reconstruction error on the computation of the weight gradients for all the n kernels incurred by the pruning. Similarly, solving this problem requires computing all the gradients in Eq.(3-12) to obtain $g_{w,j}^{l}, j \in \{1, ..., n\}$, which contradicts the goal of reducing computation. Thus, we define the importance score of each channel in δ^{l} for g_w and prune the least important ones to minimize the reconstruction error on g_w.

3.5.2.2 Importance Score

In Eq.(3-13), when a channel δ_{j}^{l} is pruned, the computation error is caused by the pruned $a^{l-1} * \delta_{j}^{l}$. Since a^{l-1} is independent of j and can be considered as a constant when measuring the importance of each channel δ_{j}^{l}, we ignore a^{l-1} and only include δ_{j}^{l} in the importance score of channel j, which is defined as follows.

$$s_j = \left\| \delta_{j}^{l} \right\|_1. \tag{3-14}$$

3.5.3 Mini-batch Pruning with Importance Score

To make the pruned channels for error propagation and gradient computation consistent with each other, we combine the importance score for these two processes. Then we scale it from instance-wise to batch-wise for mini-batch training.

The importance score s_j for gradient computation in Eq.(3-14) is a reduced form of Eq.(3-11) by setting $\gamma_1 = 0$ and $\gamma_2 = 1$. Therefore, we combine them into Eq.(3-11). Based on the per-instance importance score of each channel, we can prune channels for a mini-batch of instances to reduce the computation while maintaining accuracy. For a mini-batch of instances, we prune the same channels for all the instances. The batch-wise importance score of one channel is calculated as $S_j = \sum_{i=1}^{m} s_j^i$. m is the batch size and s_j^i is the importance score of channel j for instance i.

With the batch-wise importance score, the error map pruning process for one convolutional layer is as follows. Given a pruning ratio α, $n(1-\alpha)$ channels in the output error map δ^l need to be pruned. First, for each channel j in δ^l, we calculate the batch-wise importance score S_j. Then the importance scores of all channels are sorted and $n(1-\alpha)$ channels with the smallest S_j are marked as pruned. Then the error propagation and the computation of the weight gradients corresponding to the pruned channels are skipped to save computation.

3.5.3.1 Computation Reduction

With error map pruning, the computation cost of both the error propagation and the weight gradients is effectively reduced. With pruning ratio α, $1-\alpha$ computation in the error propagation and gradient computation is skipped, which saves about $1-\alpha$ computation in the backward pass of training. More specifically, without pruning, for one instance the computation cost of error propagation for a convolutional layer l in floating-point operations (FLOPs) is $FLOPs(\delta_{in}^l) = W_{in}H_{in}cnk_wk_h$. When pruning the number of channels in δ^l from n to αn, the computation cost is reduced to $\alpha FLOPs(\delta_{in}^l)$. For the computation of the weight gradients, before pruning the computation cost of g_w^l is $FLOPs(g_w^l) = WHcnk_wk_h$. With pruning ratio α, the cost is reduced to $\alpha FLOPs(g_w^l)$. In this way, $1-\alpha$ computation cost is reduced in the backward pass of convolutional layers.

3.5.3.2 Overhead Analysis

The computation overhead of error map pruning is negligible. It is caused by channel selection and the skipping of pruned channels. When using the ℓ_1-norm strategy in Eq.(3-11)

for the channel selection, the overhead is negligible because the sum over each kernel weight and each channel are much cheaper than the convolutional operation in the backward pass. For example, the channel selection of ResNet-110 consumes a marginal 0.53% FLOPs of the backward pass. For the overhead of skipping, since we employ structured pruning, skipping the pruned channels is simply skipping the computation involving the pruned channels, which has negligible overhead.

3.6 Experiments

We conducted extensive experiments to demonstrate the effectiveness of our approach in reducing *computation* and *energy usage*, while maintaining *accuracy* and improving *convergence speed*. We also provide a detailed *analysis* of the results. The evaluation is on six network architectures and four datasets. We first evaluate EIF and then evaluate the combined EIF+EMP approach. After that, we evaluate the practical *energy savings* on two edge devices.

3.6.1 Experimental setup

3.6.1.1 Datasets and Networks

We evaluate the proposed approaches on four datasets: CIFAR-10, CIFAR-100 [57], MNIST [61], and ImageNet [21]. We use networks with different capacities to show the scalability of the proposed approaches. The networks include large-scale networks for mobile devices and small networks for tiny sensor nodes. For large-scale networks, we employ residual networks ResNet [35] and plain networks VGG [95]. ResNet-110, ResNet-74, and VGG-16 are evaluated on CIFAR-10/100. ResNet-18 and VGG-11 are evaluated on ImageNet. For small networks, we use LeNet on MNIST.

3.6.1.2 Architectures of Instance Filter

We use different networks as the instance filter for different datasets. For CIFAR-10/100, we use ResNet-8. It has 7 convolutional layers and 1 fully-connected layer. The first layer is 3x3 convolutions with 16 filters. Then there is a stack of 3 residual blocks. The network ends with a 10/100-way fully-connected layer. For ImageNet, we use ResNet-10. It has 9 convolutional layers and 1 fully-connected layer. The first layer is 7x7 convolutions with 64 filters. Additional downsampling is conducted with a stride of 4 to reduce the computation cost. Then there is a stack of 4 residual blocks. The network ends with a 1000-way fully-connected layer. For MNIST, we use a slimmed LeNet with kernel size 3x3 and {6,16} filters for two convolutional layers.

The computation overhead of the EIF is negligible compared with the main networks. For CIFAR-10/100, the computation required for the inference of the EIF is 5.0% of ResNet-110 and 4.1% of VGG-16, respectively. The computation required for training the EIF is 5.9% of ResNet-110 and 4.8% of VGG-16, respectively. For ImageNet, the computation required for the inference and training of the EIF network is 3.4% and 3.9% of ResNet-18, and 0.81% and 1.05% of VGG-11, respectively. For MNIST, the computation required for the inference and training of the EIF is 9.5% and 7.8% of LeNet, respectively.

3.6.1.3 Training Details

We train both the main network and the instance filter simultaneously *from scratch*. For ResNet-110, ResNet-74, and VGG-16, we employ the training settings in [35]. We use SGD optimizer with momentum 0.9 and weight decay 0.0001 with batch size 128. The models are trained for 64k iterations. The initial learning rate is 0.1 and decayed by a factor of 10 at 32k and 48k iterations. For the instance filter, the learning rate is 0.1. For ResNet-18 and VGG-11, the batch size is 256 and the models are trained for 450k iterations. The learning rate is decayed by 10 at 150k and 300k iterations. For LeNet, the learning rate is 0.01 and the momentum is 0.5. The model is trained for 18.7k iterations with batch size 64. For the instance filter, the initial learning rate is 0.1 and decayed to 0.05 after 0.94k iterations.

3.6.1.4 Metrics

We evaluate the proposed approaches in two highly related but different metrics: the reduction of *computation cost* and *practical energy saving*. The computation cost is measured in FLOPs, which is a device-independent metric for computation cost [89]. The evaluation of the computation cost is conducted on NVIDIA P100 GPU with PyTorch 1.1 and measured by the THOP library [82], which will be presented in Sections 3.6.2 - 3.6.5. The second metric, practical energy saving, depends on the devices and is measured on two edge devices (NVIDIA Jetson TX2 mobile GPU and MSP432 MCU), which will be presented in Section 3.6.6.

3.6.2 Evaluating Early Instance Filtering (EIF)

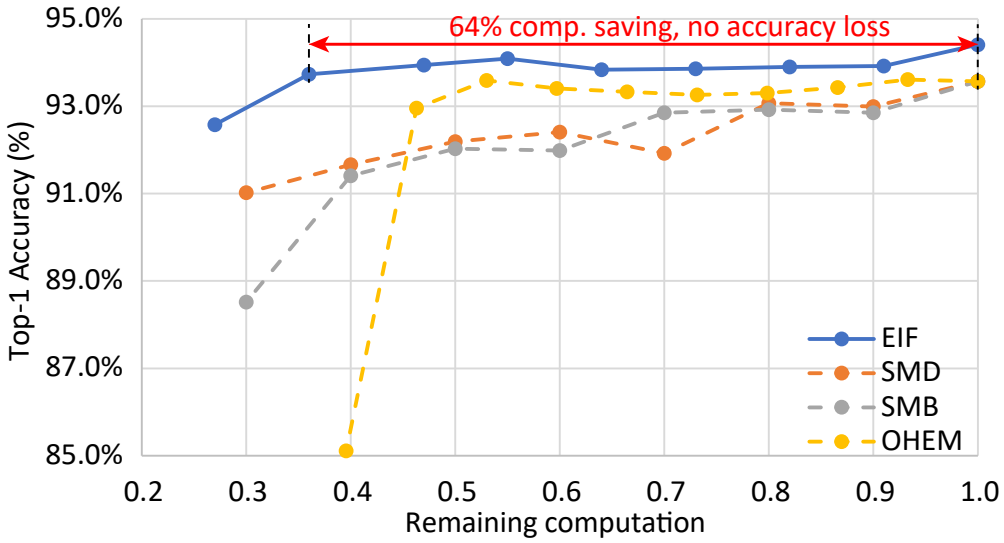


Figure 20: Top-1 accuracy by early instance filter (EIF) and baselines with ResNet-110 on CIFAR-10.

To show that the proposed early instance filtering (EIF) can effectively reduce the computation cost while maintaining or even boosting the accuracy, we compare it with two SOTA baselines and a standard training approach. Online hard example mining (OHEM) [94] selects hard examples for training by computing the loss values. Stochastic mini-

batch dropping (SMD) [105] randomly skips every mini-batch with a probability. SMB is the standard mini-batch training method by stochastic gradient descent (SGD), and the computation cost is adjusted by reducing the number of training iterations.

3.6.2.1 Computation Reduction while Boosting Accuracy

The proposed EIF substantially outperforms the baselines in terms of both accuracy and computation reduction. As shown in Figure 20, when training ResNet-110 on CIFAR-10, with different remaining computation ratios, EIF consistently outperforms the baselines by a large margin. Compared with the full accuracy by SGD (e.g. SMB with remaining computation ratio 1.0), when using only 36.50% remaining computation, EIF boosts the accuracy by 0.16% (93.73% vs. 93.57%). With only 55.45% computation, EIF boosts the accuracy by 0.52% (94.09% vs. 93.57%). Compared with SMB and SMD, under different computation ratios, EIF achieves consistently higher accuracy with a range [0.84%, 2.32%] and [0.83%, 2.28%], respectively. The significant improvement is achieved because EIF selects instances by predicting the true loss value, instead of randomly dropping the instances. Compared with OHEM, EIF consistently achieves higher accuracy with a range [0.31%, 0.98%] under different computation ratios. The improved accuracy and reduced computation cost show that the proposed instance filter effectively selects important instances for training to save computation costs.

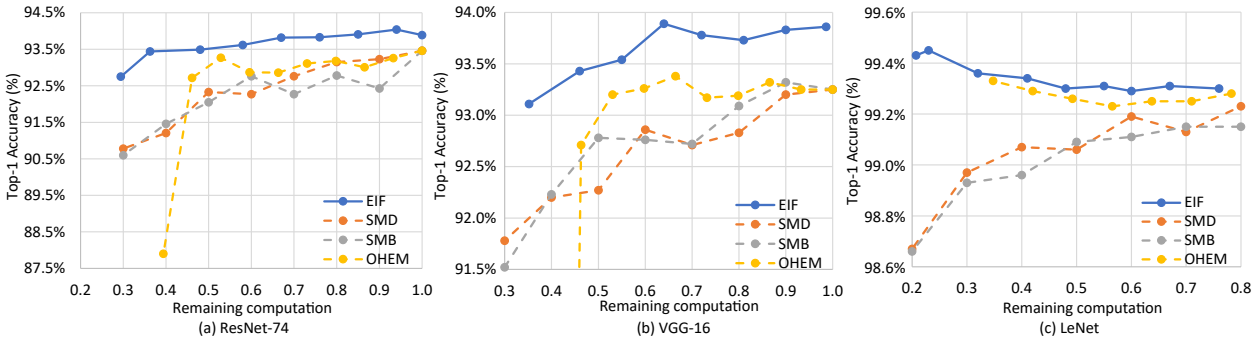


Figure 21: Top-1 accuracy by EIF and baselines with ResNet-74 and VGG-16 on CIFAR-10 and LeNet on MNIST.

To further evaluate EIF, we conduct experiments on training ResNet-74, VGG-16, and LeNet. Consistent accuracy improvement over the SOTA baselines is observed in Figure 21.

3.6.3 Evaluating EIF + EMP

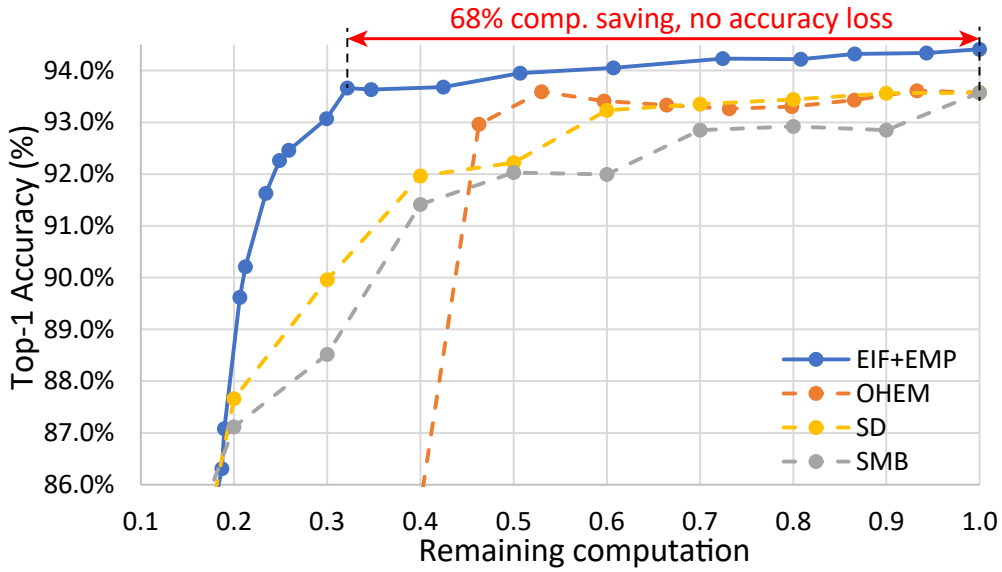


Figure 22: Accuracy of ResNet-110 on CIFAR-10 by EIF+EMP and baselines under different remaining computation ratios.

We evaluate the proposed framework EIF+EMP. Our approach effectively reduces the computation cost and achieves significantly better accuracy than SOTA baselines. Figure 22 shows the accuracy of ResNet-110 on CIFAR-10 when trained by EIF+EMP and the baselines under different remaining computation ratios. Compared with EIF or EMP only, EIF+EMP achieves more computation reduction while preserving and even boosting accuracy. With EIF only, we achieve a 63.50% computation reduction without accuracy loss. With EMP only, we achieve a 35.56% computation reduction in the backward pass without accuracy loss and a 62.22% computation reduction with a slight accuracy loss of 0.72%. By the combined EIF+EMP, with up to 67.84% computation reduction, we achieve no accuracy loss and boost the accuracy by up to 0.84% (94.41% vs. 93.57%).

We further evaluate EIF+EMP with more network architectures and datasets. Our

Table 1: The top-1 accuracy achieved by EIF+EMP using ResNet-110, ResNet-74, VGG-16 on CIFAR-10, and LeNet on MNIST.

Network	Method	Comp. Reduce	Accuracy
ResNet-110	SGD(original)	-	93.57%
	EIF+EMP	**67.84**%	**93.66**%
	OHEM [94]	60.45%	85.11%
	SD [41]	60.00%	91.96%
	EIF+EMP+Q	**95.71%**	**93.54%**
	E2Train(+Q) [105]	90.13%	91.68%
ResNet-74	SGD(original)	-	93.46%
	EIF+EMP	**63.91**%	**93.48**%
	OHEM	60.59%	87.90%
	SD	60.00%	90.99%
	EIF+EMP+Q	**95.41%**	**93.00%**
	E2Train(+Q)	90.13%	91.36%
VGG-16	SGD(original)	-	93.25%
	EIF+EMP	**67.33%**	**93.15%**
	OHEM	60.41%	71.81%
	EIF+EMP+Q	**95.54%**	**92.69%**
	E2Train(+Q)	-	-
LeNet	SGD(original)	-	99.23%
	EIF+EMP	**78.60**%	**99.47**%
	OHEM	65.24%	99.33%

approach substantially outperforms the baselines in terms of computation reduction and accuracy. We evaluate our approach with ResNet-110, ResNet-74, and VGG-16 on CIFAR-10 and LeNet on MNIST. For a fair comparison with E2Train [105], which employs quantization

[10], we use the same quantization scheme. When comparing with other baselines, we do not use quantization. The experimental results are shown in Table 1. When training ResNet-74, our approach achieves 63.91% computation savings without accuracy loss. With quantization, our approach achieves 95.41% computation saving with a marginal accuracy loss of 0.46%. E2Train achieves less computation saving of 90.13% and a much higher accuracy loss of 2.10%. Similar results are observed on ResNet-110, VGG-16, and LeNet. SD and E2Train rely on residual connections and cannot be applied to VGG-16 and LeNet.

Table 2: Top-1 accuracy by EIF+EMP and baselines with ResNet-110 and VGG-16 on CIFAR-100.

Network	Method	Comp. Reduce	Accuracy
ResNet-110	SGD (Original)	-	71.60%
	EIF+EMP	**50.02%**	**72.02%**
		56.24%	**71.63%**
	OHEM	47.01%	69.98%
	SD	50.00%	70.44%
	SMB	50.00%	67.28%
	EIF+EMP+Q	**92.92%**	**71.29%**
	E2Train(+Q)	90.13%	67.94%
VGG-16	SGD (Original)	-	71.56%
	EIF+EMP	**50.49%**	**71.59%**
		53.86%	**70.92%**
	OHEM	46.99%	65.17%
	SMB	50.00%	68.76%

3.6.3.1 Experiments on CIFAR-100

We further evaluate the proposed approaches on CIFAR-100 with ResNet-110 and VGG-16. EIF+EMP substantially outperforms the baselines in both computation reduction and

accuracy. As shown in Table 2, with ResNet-110, EIF+EMP achieves a 56.24% computation reduction while preserving the full network accuracy, and 50.02% computation reduction while boosting the accuracy by 0.42%. The baselines OHEM, SD, and SMB achieve much lower accuracy even with less computation reduction. In the case of VGG-16, EIF+EMP achieves a 50.49% computation reduction without any loss in accuracy and a 53.86% computation reduction with only a 0.64% loss in accuracy. In contrast, the baselines OHEM and SMB achieve much larger accuracy losses of 6.39% and 2.80%, respectively, with less computation reduction.

3.6.3.2 Experiments on ImageNet

We evaluate the proposed approaches on the large-scale dataset ImageNet [72]. ImageNet consists of 1.2M training images in 1000 classes. The main networks are ResNet-18 and VGG-11.

The proposed EIF+EMP effectively reduces the computation cost in training while preserving the accuracy on the large-scale dataset. As shown in Table 3, when training ResNet-18, with a 58.91% computation reduction in training, EIF+EMP boosts the top-1 accuracy by 0.51% (70.27% vs. 69.76%) and boosts the top-5 accuracy by 0.55%. With a more aggressive computation reduction of 64.71%, EIF+EMP still boosts the top-5 accuracy by 0.27% (89.35% vs. 89.08%). EIF+EMP consistently outperforms the SOTA baselines by a large margin. Similar results are observed on VGG-11.

3.6.4 Convergence Speed

The proposed approaches improve the convergence speed in the training process. The test error (i.e. 100% - accuracy on the test dataset) over the computation cost during training is shown in Figure 23. The proposed EIF, EMP and combined EIF+EMP approaches converge faster than the baselines, represented as lower test error (higher accuracy) with the same computation cost. More specifically, EIF+EMP achieves 3.1x faster convergence and 0.09% accuracy improvement compared with the standard mini-batch approach (SMB). The SOTA baselines OHEM and SD achieve lower convergence speed and larger accuracy loss of 8.46%

Table 3: Top-1 and Top-5 accuracy by EIF+EMP and baselines with ResNet-18 and VGG-11 on ImageNet.

Network	Method	Comp. Reduce	Acc. (top-1)	Acc. (top-5)
ResNet-18	SGD (Original)	-	69.76%	89.08%
	EIF+EMP	**58.91%**	**70.27%**	**89.63%**
		64.71%	**68.98%**	**89.35%**
	OHEM	46.67%	62.09%	87.08%
	SD	50.00%	65.36%	86.41%
	SMB	50.00%	65.94%	87.50%
VGG-11	SGD (Original)	-	70.38%	89.81%
	EIF+EMP	**51.63%**	**70.36%**	**89.98%**
		60.59%	**70.01%**	**89.83%**
	OHEM	46.59%	56.39%	85.62%
	SMB	50.00%	63.76%	86.49%

and 1.61%, respectively.

3.6.5 Quantitative and Qualitative Analysis

3.6.5.1 Effectiveness of Adaptive Loss Threshold

The proposed early instance filter effectively predicts a pre-defined percentage of input instances as high-loss and the adaptive loss threshold effectively adjusts the loss threshold as the labeling strategy to train the filter. In Figure 24(a), the pre-defined high loss ratio is 40% for training ResNet-110 on CIFAR-10. The number of predicted high-loss instances, averaged every 390 iterations, is stabilized at about 51, which effectively selects 40% high-loss instances

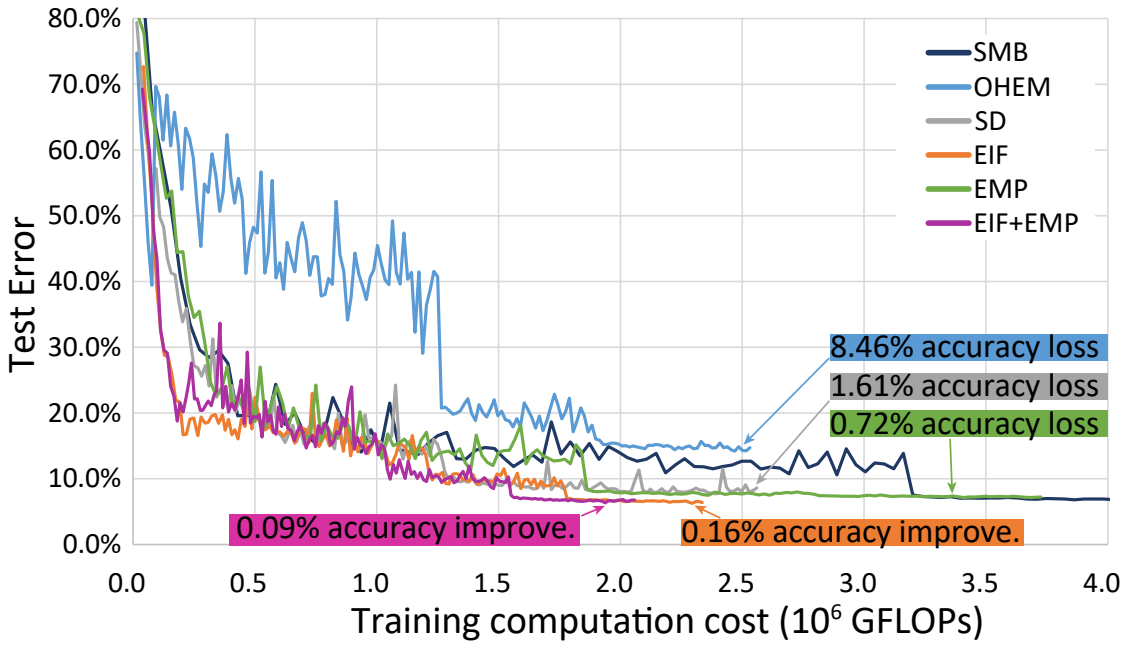


Figure 23: Convergence speed of ResNet-110 on CIFAR-10 during training with different approaches.

on average from 128 instances in each mini-batch. We further compare the proposed adaptive loss threshold with the static loss threshold. The static loss threshold is set to 1.0 in Figure 24(b). The goal of training is to minimize the loss. However, the static loss threshold cannot effectively decrease the loss of the main model as shown in the blue line, and results in low accuracy. This is because the static loss threshold cannot track the latest state of the main model. Therefore, it cannot effectively stabilize the number of predicted high-loss instances to train the main model.

3.6.5.2 Effectiveness of Weighted Loss for Training EIF

The weighted loss in Eq.(3-7) effectively trains the EIF network to make accurate loss predictions, which eventually results in higher accuracy of the main model. As shown in Figure 25, when the weighted loss is employed, the wrong loss prediction ratio by the EIF is

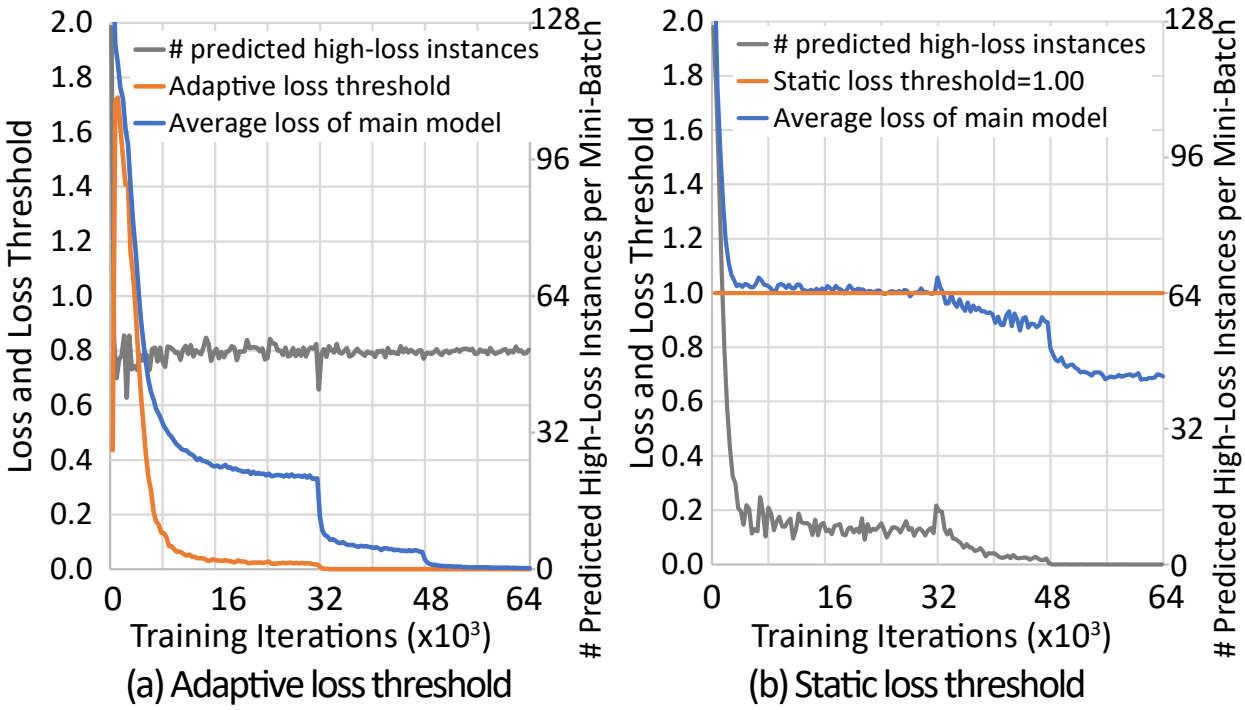


Figure 24: The adaptive loss threshold (left) tracks the state of the main model and stabilizes the number of preserved instances with predicted high loss by EIF.

much lower than that without weighted loss. The pre-defined high-loss ratio is 30%, and the corresponding low-loss ratio is 70%. When the weighted loss is used for training EIF, the average wrong loss prediction ratio by EIF is reduced from 20.31% to 8.59%. This accurate loss prediction effectively selects high-loss instances to train the main model and results in significantly higher accuracy of the main model, which is 94.05% with weighted loss vs. 90.58% without weighted loss.

3.6.5.3 Overhead of EIF

The proposed early instance filter has marginal energy and computation overhead. The average energy and computation overhead of the EIF network per training iteration (e.g. one mini-batch of 128 instances) when training ResNet-110 on the CIFAR-10 dataset is shown in Figure 26. As shown in the yellow bar in Figure 26(a), the energy overhead of the EIF

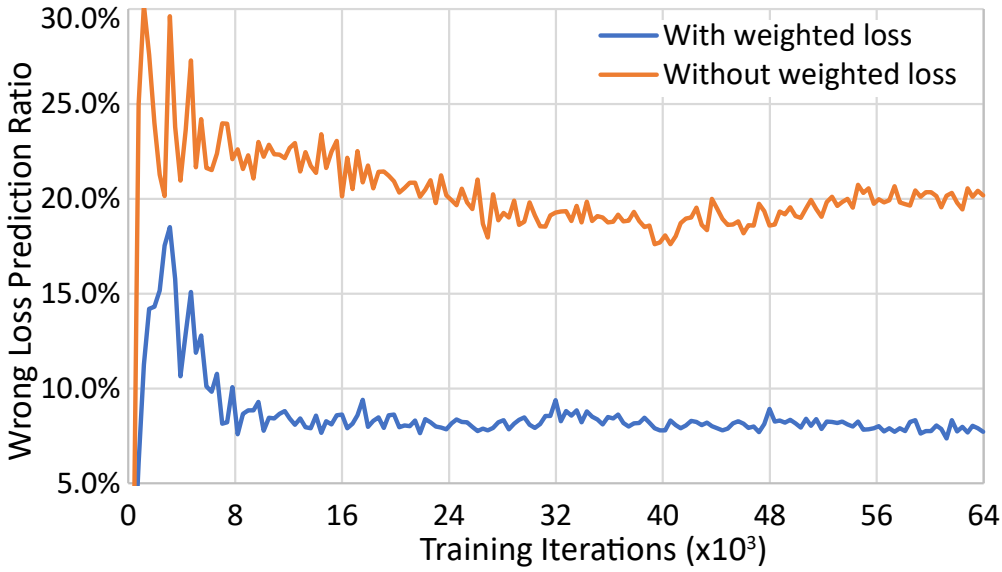


Figure 25: Incorrect loss prediction ratio of EIF with and without weighted loss.

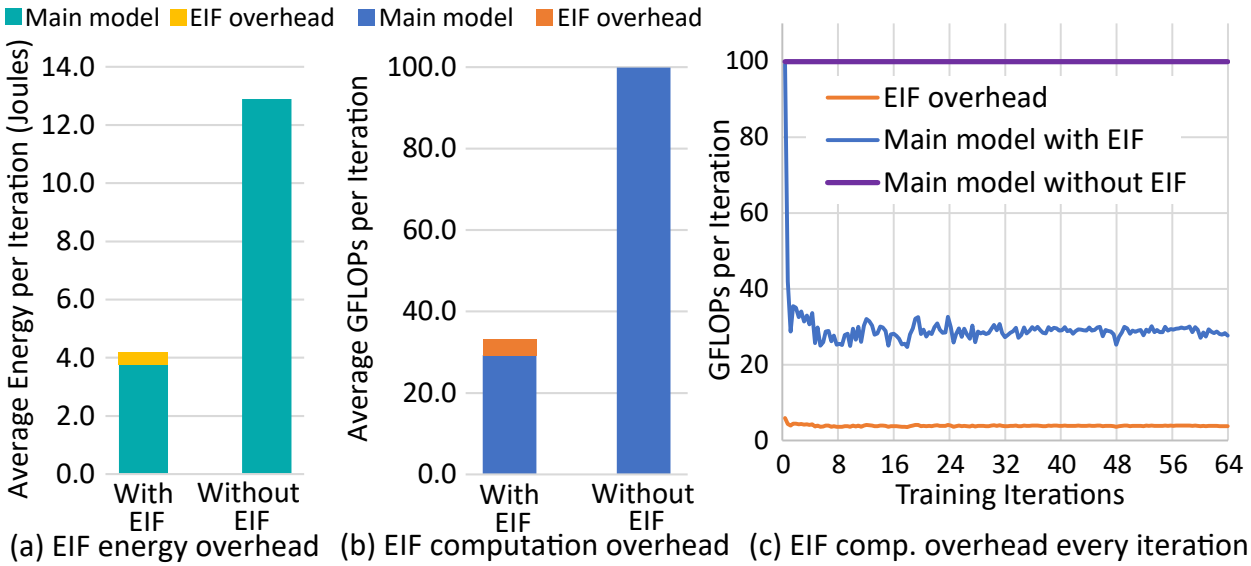


Figure 26: Energy and computation overhead of EIF. Energy overhead is measured on NVIDIA Jetson TX2 mobile GPU.

network (measured on NVIDIA Jetson TX2) is 0.43J per iteration, which is 10.22% of the total energy cost 4.18J when training with EIF. Without EIF, the energy cost is 12.90J per iteration. As shown in Figure 26(b), the computation overhead of EIF is 3.88 GFLOPs, which is 11.65% of the total computation cost of 33.21 GFLOPs when training with EIF. Without EIF, the computation cost is 99.91 GFLOPs per iteration. The detailed EIF computation overhead across all training iterations is shown in Figure 26(c). While the overhead of EIF is not zero, the proposed approach achieves 67.60% energy saving and 66.76% computation saving while fully preserving the accuracy.

Figure 27: Preserved and dropped instances by EIF when training ResNet-110 on CIFAR-10 and LeNet on MINST.

Preserved and Dropped Instances by EIF. To better understand the instances selected by the early instance filter, we cluster the instances that the filter preserves and drops when training ResNet-110 on CIFAR-10 and LeNet on MNIST, as shown in Figure 27. We find that the dropped instances show the full objects with typical characteristics. The preserved instances either only show part of the object or show non-typical characteristics, even hard for humans to understand. This result shows the early instance filter can effectively find important instances to train the network.

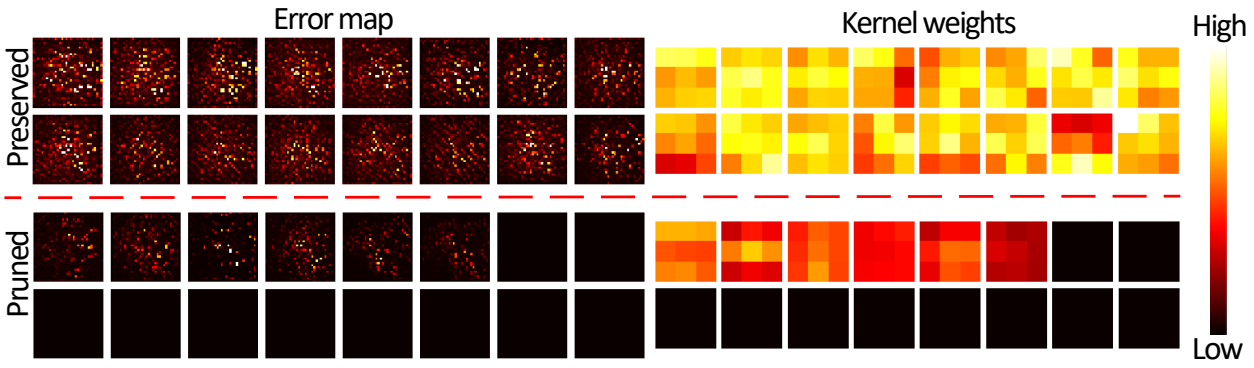


Figure 28: Visualization of the pruned and preserved channels in the error map and corresponding convolutional kernels.

3.6.5.4 Analysis of Error Map Pruning

To better understand the pruned and preserved channels in the backward pass by using error map pruning, we visualize them to analyze the effectiveness of the proposed channel selection approach. The preserved and pruned channels in the error map and corresponding kernel weights in the conv2 layer of VGG-16 are shown in Figure 28. The pruned channels are darker with smaller values than the preserved channels, which are brighter with larger values. Therefore, the pruned channels will have the least influence on both the error propagation and computation of weight gradients. This result shows the proposed error map pruning approach effectively selects the channels to prune to minimize the influence on training.

3.6.6 Practical Energy Saving on Hardware Platforms

The energy cost of training consists of both the computation cost and the memory access cost. While the former dominates the energy cost and is represented by the commonly used metric FLOPs [89], the *energy saving* ratio can be slightly different from the *computation reduction* ratio. To evaluate the energy saving, we conduct extensive experiments on two edge platforms and evaluate the proposed approaches in terms of *practical energy saving* and *accuracy*.

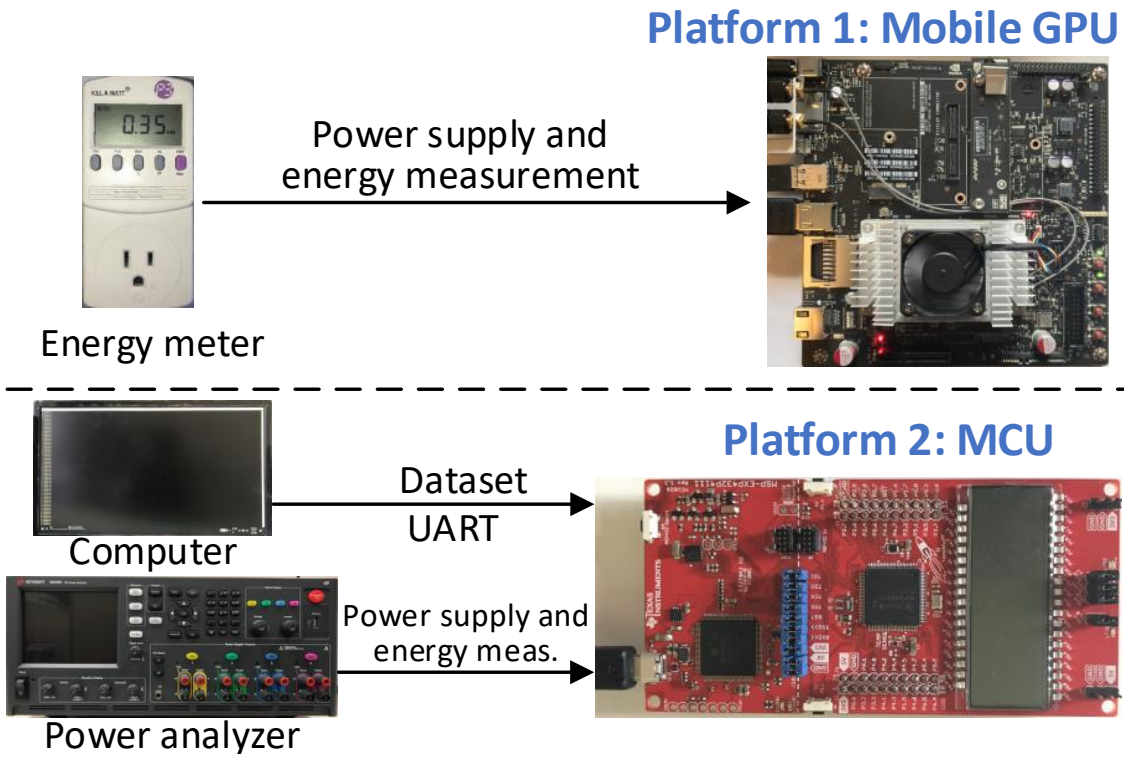


Figure 29: Energy measurement setup for training on two edge platforms, including mobile-level devices (top) and sensor node-level devices (bottom).

3.6.6.1 Hardware Setup

We apply the proposed training approach on two edge platforms to evaluate *realistic energy saving*. **For mobile-level devices,** we train ResNet-110, ResNet-74, and VGG-16 on an NVIDIA Jetson TX2 mobile GPU with CIFAR-10 and CIFAR-100 datasets by PyTorch 1.1. We use an energy meter to measure the energy cost as shown at the top of Figure 29. **For sensor node-level devices,** we train LeNet on the MSP432 MCU. We use C language to implement the training process. Since the MCU cannot store the entire dataset, we use a computer to feed the training data into the MCU via UART in the training process. We use the Keysight N6705C power analyzer to measure the energy cost.

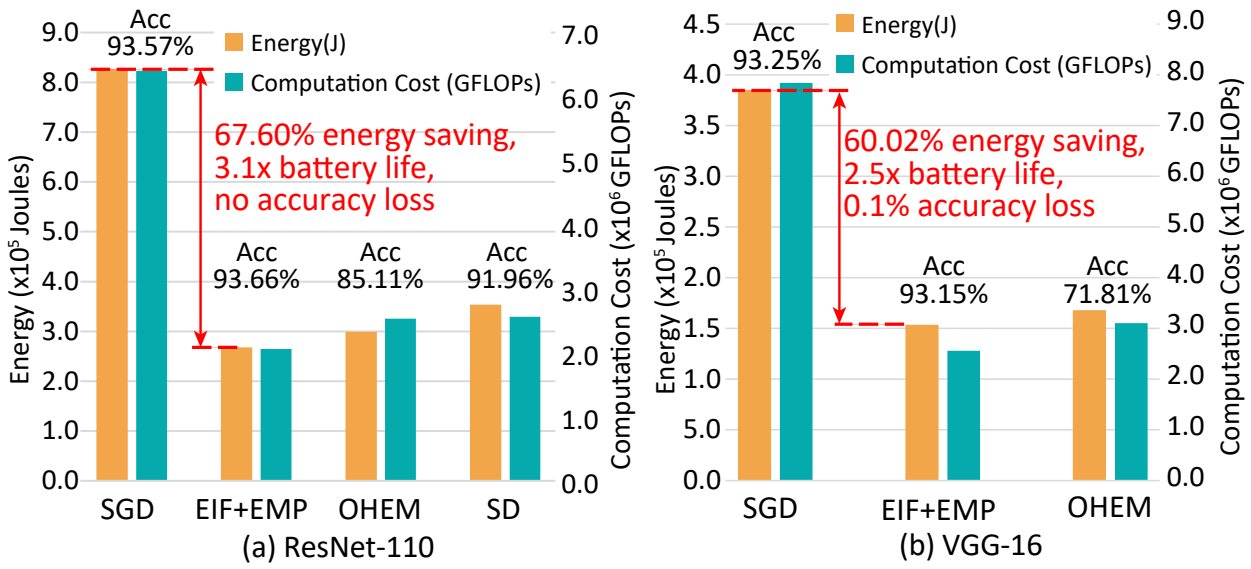


Figure 30: Energy saving when training ResNet-110 and VGG-16 on Nvidia Jetson TX2 mobile GPU with CIFAR-10 dataset.

3.6.6.2 Energy Saving of Training on Mobile GPU

We evaluate the energy saving by EIF+EMP on mobile-level devices. We repeat all the experiments in Table 1 and Table 2 on the mobile GPU to measure the practical energy saving, except for the LeNet, which will be evaluated on MCU. Our approach effectively reduces the energy cost of on-device training. Compared with the original SGD, the proposed EIF+EMP achieves energy savings of 67.60%, 63.57%, and 60.02% in the training of ResNet-110, ResNet-74, and VGG-16 on CIFAR-10, respectively, as shown in Figure 30 (the result of ResNet-74 is not shown for conciseness). The energy savings prolong battery life by 3.1x, 2.7x, and 2.5x while improving the accuracy or incurring a slight 0.1% accuracy loss. Compared with the SOTA baselines OHEM and SD, our approach achieves significantly higher accuracy when similar energy saving is achieved. SD relies on residual connections and cannot be applied to VGG-16. Besides, the practical energy-saving ratios are very close to the computation reduction ratios represented by FLOPs, which shows the computation reduction in FLOPs can generalize well to energy-saving on hardware platforms. Similar results are observed

on the CIFAR-100, on which we achieve 54.22% and 46.64% energy saving (2.2x and 1.9x battery life) for ResNet-110 and VGG-16 without any accuracy loss, respectively.

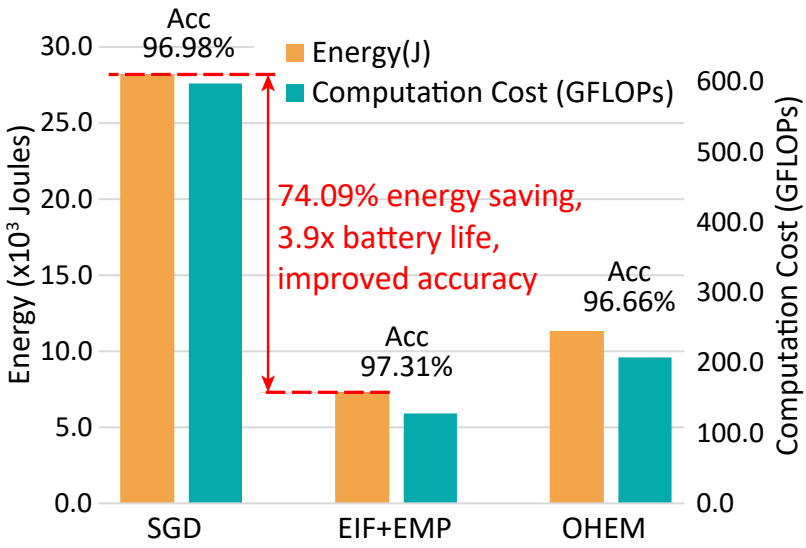


Figure 31: Energy saving when training LeNet on MSP432 MCU. EIF+EMP prolongs 3.9x battery life.

3.6.6.3 Energy Saving of Training on MCU

We evaluate the energy saving by EIF+EMP on sensor node-level devices (i.e. MCUs). We train LeNet on MCU MSP432 for one epoch including 60000 instances and measure the energy cost and accuracy. Due to the limited runtime memory, we set the batch size to 1. EIF+EMP significantly reduces the energy cost of training on MCUs and effectively prolongs battery life. As shown in Figure 31, when training LeNet on MSP432 MCU, EIF+EMP effectively reduces the energy cost by 74.09% while improving the accuracy by 0.33%. This prolongs battery life by 3.9x. OHEM, while not fully feasible on MCU, achieves a much lower energy saving of 59.78% with an accuracy loss of 0.32%. This result shows EIF+EMP greatly improves the battery life of tiny sensor nodes and outperforms the baselines.

3.7 Summary

This work aims to enable on-device training of convolutional neural networks by reducing the computation cost at training time. We propose two complementary approaches to reduce the computation cost: early instance filtering (EIF), which selects important instances for training the network and drops trivial ones, and error map pruning (EMP), which prunes insignificant channels in the error map in back-propagation. Experimental results show superior computation reduction with higher accuracy compared with SOTA techniques.

4.0 Unsupervised On-device Representation Learning

This chapter presents a project that aims to enable on-device contrastive learning from streaming input data by selecting the most representative data [111]. The chapter is structured as follows. First, the motivation is presented. Next, the background and related works in the field are introduced. Then, the details of the proposed unsupervised data selection technique from unlabeled data stream to improve self-supervised on-device learning and corresponding optimizations are presented. After that, the experimental results are shown to demonstrate the effectiveness of the proposed techniques for self-supervised on-device learning. Finally, a video demo of the project is described and the conclusion is presented to summarize this project.

4.1 Introduction

The deployment of deep learning models on edge devices has become increasingly common for various tasks, including search and rescue robots [91] and wildfire surveillance UAVs [88]. The initial model training is typically performed on high-performance servers and subsequently deployed on these devices without any further training. However, it is often beneficial for these devices to learn from real-world input data, such as images captured by a camera, either based on a pre-trained model or starting from scratch when deployed in an unknown environment [81]. This enables the model on robots or UAVs to adapt to new environments [92].

Although it is possible to send a small amount of data to servers for labeling, sending all new data is often impractical due to the need for expert knowledge, concerns about data privacy, high communication costs, and potential latency issues [13]. Therefore, rather than relying on traditional server-based training using fully labeled datasets, it is desirable to perform in-situ learning on streaming data using as few labels as possible.

Contrastive learning, as an effective self-supervised learning approach [34], can learn visual representations from unlabeled data to improve the feature extractor (convolutional layers)

in the model. After contrastive learning, the classifier (fully connected layers) can be trained on top of the improved feature extractor by using a few labeled data to achieve improved classification performance. Contrastive learning is conventionally conducted by using a large dataset, which is completely collected before the training starts. In the learning process, each mini-batch is randomly sampled from the whole dataset to update the model [18]. On edge platforms such as robots and UAVs, the data are collected by sensors such as cameras and continuously fed into the device. While it is theoretically possible to store the constantly generated massive unlabeled data on the device and employ contrastive learning, both the storage and energy overhead associated with writing and reading these data from storage devices (e.g. Flash memory) can be prohibitive in practice.

To learn from the unlabeled data stream without accumulating a large dataset, a small data buffer can be used to form each mini-batch for training. Existing contrastive learning frameworks [18, 34] assume that each mini-batch is independent and identically distributed (iid) by sampling uniformly at random from all the classes (i.e. each class has representative data in this mini-batch). However, it is challenging to maintain the most representative data in the buffer such that learning from this buffer will efficiently reach an accurate model due to the following two reasons. *First*, the streaming data collected on edge devices are usually temporally correlated [79] and result in a correlation within each mini-batch. This is because a long sequence of data in the temporally correlated stream can be in the same class [32]. For example, in wildlife monitoring, goats from a group can appear in adjacent images captured by a continuous monitoring camera [43] at some time, while zebras can appear in adjacent images at another time. *Second*, there is no easy way to select representative data for each class from the non-iid streaming data due to the fact that the streaming data are *unlabeled*. If labels were available for all the data, we could easily select representative data for each class [32] based on all the labels even if the streaming data is non-iid. Without addressing this challenge, directly learning from these temporally correlated non-iid mini-batches will result in slow learning speed and poor learned representations.

To improve the accuracy and expedite the learning process, it is essential to maintain a data buffer filled with representative data from the streaming data. To achieve this goal, this paper defines a contrast score, which is computed by the similarity between the features of

data and its flipped view. The contrast score of each data measures the quality of feature representation encoded by the model. Based on the contrast score, we propose a data replacement policy to maintain a representative data buffer. Data with a low-quality encoded representation by the model is more valuable for learning since they have not been effectively learned. These data will be maintained in the buffer for further learning. On the other hand, data with high-quality representations have been effectively learned, and they will be dropped to save places for more valuable data. After contrastive learning effectively learns from the unlabeled data and improves the feature extractor, the classifier needs to be updated as well. Since training the classifier without any labels does not generate meaningful accuracy, we will send as little as 1% of the data to the server for labeling to improve the classifier and overall accuracy.

In summary, the main contributions of the paper include:

- **Self-supervised on-device learning framework.** We propose a framework to form mini-batches of training data for self-supervised contrastive learning on-the-fly from the unlabeled input stream. It only uses a small data buffer and eliminates the necessity of storing all the streaming data on the device.
- **Contrast scoring for data selection.** We propose a data replacement policy by contrast scoring to maintain the most representative data in the buffer for on-device contrastive learning. Labels are not needed in the data replacement process, and the selected data will generate large gradients that benefit the learning most.
- **Lazy scoring for reduced computation overhead.** We propose a lazy scoring strategy to reduce the runtime overhead of data scoring. The data scores are updated every several iterations instead of every iteration to save computation.

Experimental results on multiple datasets including CIFAR-10, CIFAR-100, SVHN, ImageNet-20, ImageNet-50, and ImageNet-100 show that the proposed framework achieves significantly higher accuracy than the SOTA techniques and greatly improves the learning speed. With 1% labeled data on the CIFAR-10 dataset, the proposed framework achieves 28.36% higher accuracy than using the 1% labeled data for direct supervised learning. The proposed data selection method based on contrast scoring achieves 13.9% higher accuracy

than the current SOTA approach [49]. At the same level of accuracy, the proposed approach enables 2.67x faster learning compared to the baseline.

4.2 Background and Related Work

4.2.1 Background of Contrastive Learning

Contrastive learning is a self-supervised approach to learning an encoder (feature extractor) for extracting visual representations from the input image [11, 16, 18, 29, 33, 34, 112, 121]. In this work, we employ the contrastive learning approach from [18] since it performs on par with its supervised counterpart. For an input image x, its representation vector h is obtained by $h = f(x)$, where $f(\cdot)$ is the backbone of a deep learning model (i.e. convolutional layers). To boost the performance of learned representation, a project head $g(\cdot)$ is used to map the data representation to the latent space as a vector $z = g(h) = g(f(x))$ where contrastive loss is applied. To create a positive pair (z_i, z_{i^+}), one input x is augmented twice as (x_i, x_{i^+}) and then fed into the encoder to get representation vectors $(h_i, h_{i^+}) = (f(x_i), f(x_{i^+}))$, which are further projected by $g(\cdot)$ and normalized as (z_i, z_{i^+}). Then for each positive pair (z_i, z_{i^+}) in one mini-batch, the contrastive loss is applied to compute the loss ℓ_{i,i^+} as follows:

$$\ell_{i,i^+} = -\log \frac{\exp(z_i \cdot z_{i^+}/\tau)}{\exp(z_i \cdot z_{i^+}/\tau) + \sum_{i^-} \exp(z_i \cdot z_{i^-}/\tau)}, \tag{4-1}$$

where z_{i^-} is the representation vector of other data (serving as negatives to contrast with) in the same mini-batch, and τ is the temperature. Minimizing $\sum \ell_{i,i^+}$ in one mini-batch by iteratively updating the model will learn an encoder to generate representations.

4.2.2 Related Work

4.2.2.1 Contrastive Visual Representation Learning

Existing works [18, 34] employ contrastive loss for representation learning and achieve high accuracy on classification and segmentation tasks. [54, 79] use the temporal correlations

in the streaming data to improve representation learning. However, all these works assume that the whole training dataset is available in the learning process, and each mini-batch can be formed by sampling from the dataset. Each mini-batch consists of independent and identically distributed (iid) data. But when learning from the streaming data, which cannot be assumed to be iid on edge devices, the data is collected sequentially as it is. Besides, random sampling from the entire input stream to create iid mini-batches is infeasible since it requires storing all the data. Therefore, an approach to form mini-batches on-the-fly while including the most representative data in each mini-batch is needed to enable efficient and accurate on-device contrastive learning.

4.2.2.2 Data Selection in Streaming and Continual Learning

There are several supervised streaming and continual learning models that can learn from a stream of data [8]. To overcome the problem of catastrophic forgetting of previously seen data, a data buffer is usually needed to store previous data for rehearsal [8, 14, 32]. The main drawback of these approaches is that *data labels* are needed to maintain the buffer. However, labeling all the data in the streaming is prohibitive or even infeasible on edge devices. Therefore, existing methods cannot be applied directly to contrastive learning and an effective data selection approach that works on *unlabeled* data is needed.

4.3 Self-Supervised On-Device Learning by Selective Data Contrast

This paper proposes a framework to efficiently learn data representations from the unlabeled input stream on-the-fly without accumulating a large dataset due to storage limitations on edge devices. To maintain the most representative data in the buffer such that learning from these data will benefit the model most, we propose a data replacement policy based on *Contrast Score* by measuring the quality of representation for each data without using labels. Data with low-quality representations have not been effectively learned by the model and will be maintained in the buffer for further learning, while data with high-quality

representations will be dropped. The contrast scoring is supported by the theoretical analysis that data with higher scores will generate larger gradients and accelerate the learning process.

In this section, we will first present the framework overview in Section 4.3.1. Then we will introduce the proposed contrast scoring for data selection in Section 4.3.2. After that, we will theoretically analyze the effectiveness of contrast scoring in Section 4.3.3. Finally, we will introduce lazy scoring to reduce the runtime overhead of contrast scoring in Section 4.3.4.

4.3.1 Framework Overview

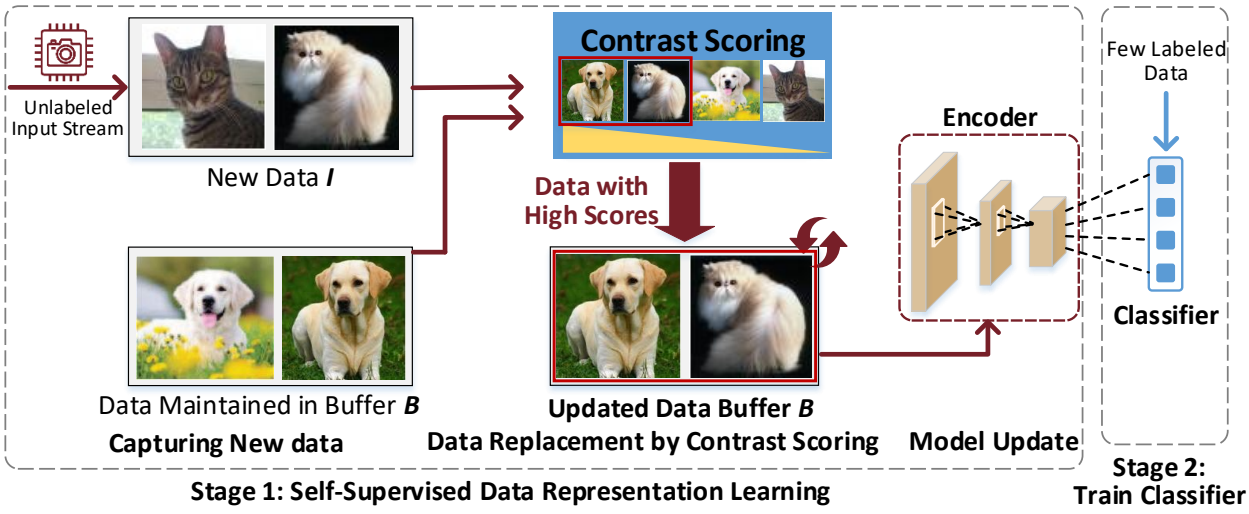


Figure 32: Overview of on-device contrastive learning framework.

As shown in Figure 32, the proposed framework has two stages. The first stage learns an encoder (i.e. convolutional layers) by self-supervised contrastive learning to generate data representations (i.e. low-dimensional vectors) from the high-dimensional unlabeled inputs (e.g. images). The second stage learns a classifier by using a few (e.g. 1%) labeled data on top of the learned representations.

In stage 1, the proposed framework consumes the input streaming data on-the-fly to update the model for improved representation. We only use a small data buffer B (i.e. the same size as one mini-batch) to maintain the most representative data. When a segment of new input I arrives, both the new data in I and the data in buffer B will be scored to

find the most representative data. While any size of I can be used, for simplicity we assume I has the size as B by setting size(I) = size(B). Then the data with the highest scores in $B \cup I$ will be selected and put into B. In this way, the data replacement process always maintains the most representative data among the new and the old ones. After each iteration of data replacement, the data preserved in the data buffer B will serve as one mini-batch for updating the model once. The detailed data replacement policy will be described in the next subsection.

4.3.2 Data Replacement By Contrast Scoring

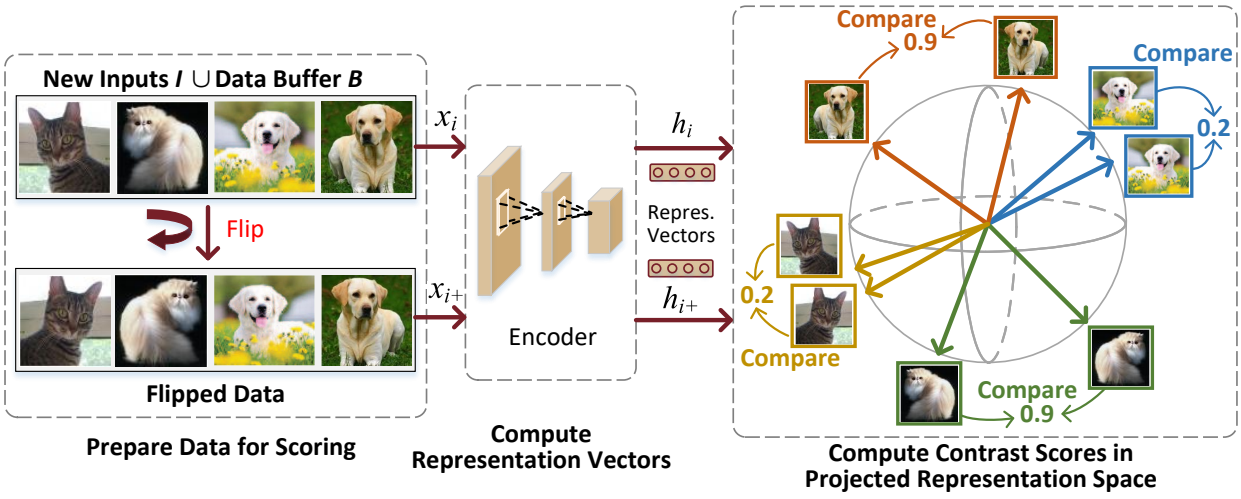


Figure 33: Contrast scoring for data replacement.

4.3.2.1 Contrast Scoring

For each input x_i, the *contrast scoring* function $S(x_i)$ aims to measure the quality of the representation vector $h_i = f(x_i)$ generated by the base encoder $f(\cdot)$. Intuitively, if the representation of x_i is not good, x_i will be valuable data for updating the base encoder since it can still learn from x_i to improve its capability of encoding x_i. To achieve this, as shown in Figure 33, for each image x_i from the input stream and the buffer, we generate another view x_{i+} by horizontal flipping. Then we feed both x_i and x_{i+} into the encoder and generate

the representation vectors h_i and h_{i^+} for these two views. Ideally, if the encoder has learned to generate effective representations of x_i, h_i and h_{i^+} will be identical or very similar. After that, based on h_i and h_{i^+}, the score for x_i is computed by the contrast scoring function $S(\cdot)$.

The contrast scoring function $S(\cdot)$ is defined as:

$$\begin{aligned} S(x_i) = dissim(x_i, x_{i^+}) &= 1 - similarity(z_i, z_{i^+}) \\ &= 1 - z_i^T z_{i^+}, \quad x_i \in \{B \cup I\}, \end{aligned} \tag{4-2}$$

$$\text{where } z_i = g(h_i)/\|g(h_i)\|_{\ell_2},\ z_{i^+} = g(h_{i^+})/\|g(h_{i^+})\|_{\ell_2}, \tag{4-3}$$

where h_i and h_{i^+} are the representation vectors generated by the base encoder $f(\cdot)$ as $h_i = f(x_i)$ and $h_{i^+} = f(x_{i^+})$, taking data x_i and its horizontally flipped view x_{i^+} as inputs, respectively. z_i and z_{i^+} are ℓ_2-normalized vectors from the projection head $g(\cdot)$ to enforce $\|z_i\|_{\ell_2} = \|z_{i^+}\|_{\ell_2} = 1$. In this way, the dot product $z_i^T z_{i^+}$ is in the range [-1,1], and $S(x_i)$ is non-negative and in the range [0,2].

The contrast scoring function Eq.(4-2) measures the dissimilarity between the projected representation vectors of an image x_i and its horizontal flip x_{i^+}, where *a higher score means a larger dissimilarity*. Essentially, the representation of one image needs to be invariant to image transformations [47], and the representations of x_i and x_{i^+} need to be as similar as possible. Since a higher score represents a larger dissimilarity and less invariance, input x_i with a higher score is more valuable for updating the base encoder because the base encoder still cannot generate sufficiently good representations of it. By updating the encoder with x_i using the contrastive loss [18], which aims to maximize the similarity of two strongly augmented views of x_i, the score of x_i in Eq.(4-2) will decrease and x_i will have a lower probability of being selected into the next mini-batch in Eq.(4-4). In this way, more valuable data to update the base encoder will have a higher probability of being selected for the next mini-batch and others are more likely to be dropped. A detailed analysis of the effectiveness of contrast scoring will be provided in Section 4.3.3.

4.3.2.2 Contrast Score Design Principle

Contrast scoring is a metric to represent the capability of the base encoder in generating the representation $h_i = f(x_i)$ for x_i. Thus, it should only relate to the image itself and the encoder. In Figure 33, when generating a pair of inputs (x_i, x_{i^+}) to $S(\cdot)$ from an image x_i, we find it crucial to avoid any randomness (e.g. random crop) and *only apply the weak data augmentation* (i.e. horizontal flipping) to generate x_{i^+}. The reason is that this weak augmentation is deterministic and provides consistent inputs to $S(\cdot)$. In this way, the score $S(\cdot)$ is deterministic to x_i and is consistent in different runs of $S(\cdot)$.

4.3.2.3 Contrast Score Based Data Selection

The objective at iteration t is to construct the subsequent mini-batch B_{t+1} by selecting the most informative data from the new data I_t and the buffer B_t. This selection is performed with the aim of maximizing the model's learning from B_{t+1}, resulting in the most significant improvement in performance. To achieve this, we apply the contrast scoring function $S(\cdot)$ to both the data already in the buffer B_t and new data I_t. B_{t+1} is formed by selecting the data with the highest contrast scores in $B_t \cup I_t$:

$$B_{t+1} = \{x_i | x_i \in B_t \cup I_t, i \in topN(\{S(x_i)\}_{i=1}^{2N})\}, \tag{4-4}$$

where $topN()$ returns the indices of x_i with the top N scores. In this way, the most representative data is maintained in the buffer by using the proposed contrast scoring.

4.3.3 Effectiveness of Contrast Score

The proposed contrast scoring effectively selects data that can generate large gradients, which benefits the learning most. To understand this, for each data x_i in one mini-batch, the gradient of contrastive loss ℓ_{i,i^+} in Eq.(4-1) with respect to the representation vector z_i is computed as:

$$\frac{\partial \ell_{i,i^+}}{\partial z_i} = -\frac{1}{\tau}\left(\left(1 - p_{z_{i^+}}\right) \cdot z_i - \sum_{z_{i^-}} p_{z_{i^-}} \cdot z_{i^-}\right), \tag{4-5}$$

$$p_z = \frac{\exp\left({z_i}^T z/\tau\right)}{\sum_{z_j \in \{z_{i^+}, z_{i^-}\}} \exp\left({z_i}^T z_j/\tau\right)}, \quad z \in \{z_{i^+}, z_{i^-}\}, \tag{4-6}$$

where p_z is the probability distribution generated by applying the softmax function to the similarity ${z_i}^T z_j$ between z_i and each $z_j \in \{z_{i^+}, z_{i^-}\}$ in the mini-batch. For $z = z_{i^+}$, $p_{z_{i^+}}$ is the matching probability of z_i with its positive pair z_{i^+}. Similarly, for $z = z_{i^-}$, $p_{z_{i^-}}$ is the matching probability of z_i with a negative pair z_{i^-} (i.e. the representation vector of other data in the same mini-batch).

Data with a small contrast score $S(x_i)$ generates a near-zero gradient and contributes almost nothing to the learning process. On the other hand, a data x_i with a high contrast score $S(x_i)$ in Eq.(4-2) corresponds to a large gradient in Eq.(4-5), which contributes much to the learning process. To understand this, we analyze the relationship between the contrast score in Eq.(4-2) and the gradient in Eq.(4-5) in two cases:

Case 1: Data with a small contrast score generates a near-zero gradient. A small contrast score in Eq.(4-2) corresponds to a large similarity between z_i and z_{i^+}. Therefore, the value of dot product ${z_i}^T z_{i^+}$ will be large and dominate the elements in the softmax function in Eq.(4-6). As a result, $p_{z_{i^+}}$ will be large and near 1. Since $p_{z_i^+} + \sum_{z \in z_{i^-}} p_z = 1$ as a property of the softmax function, the values of all $p_{z_{i^-}}$ will be small and near 0. In this way, $1 - p_{z_{i^+}}$ as well as all $p_{z_{i^-}}$ will be near 0, and the gradient $\frac{\partial \ell_{i,i^+}}{\partial z_i}$ will be near 0. Using the near-zero gradient to perform one gradient descent step $w \leftarrow w - \eta \frac{\partial \ell_{i,i^+}}{\partial z_i}$ does not contribute to the learning since there is almost no change in the weight.

Case 2: Data with a high contrast score generates a large gradient. When the contrast score is high, z_i and z_{i^+} are dissimilar to each other. By applying the same reasoning as case 1, $1 - p_{z_{i^+}}$ and all $p_{z_{i^-}}$ will be near 1, and the gradient $\frac{\partial \ell_{i,i^+}}{\partial z_i}$ will be large, which significantly contributes to the learning process.

Therefore, by using the proposed contrast score, trivial data that only generate near-zero gradients will be dropped while important data that can generate large gradients will be maintained in the buffer for learning.

4.3.4 Lazy Scoring

Computing the scores for new data and data in the buffer requires feeding these data into the base encoder to generate the representations. This computation incurs additional time overhead. To minimize the overhead, we propose lazy scoring, in which part of the data scores can be reused to reduce computation.

We made the following two observations as the foundation of lazy scoring. *First*, during each iteration of data replacement, most of the data (i.e. about 90%) in the buffer are preserved while most of the new data are directly dropped. Therefore, by reusing contrast scores of data in the buffer, a large portion of the computation in scoring can be reduced. *Second*, the score $S(x_i)$ of data x_i only slightly changes across several adjacent iterations. This is because the score of data x_i only depends on itself and the base encoder $f(\cdot)$. x_i remains constant and $f(\cdot)$ is slowly updated across iterations. Therefore, $S(x_i)$ is only slowly updated following the pace of $f(\cdot)$, and the score $S(x_i)$ computed iterations ago still provides meaningful information of x_i.

To achieve lazy scoring, as long as data x_i remains in buffer B, its score is updated in every T iteration instead of in every iteration. More specifically, for each x_i in B, we track its age $age(x_i)$ in the number of iterations since it was placed in B. When performing scoring, we separate data in B into two subsets, in which one needs scoring while the other does not. The subset of data that needs scoring is denoted as:

$$B'_t = \{x_i \mid x \in B_t, age(x_i) \ mod \ T = 0\}. \tag{4-7}$$

When scoring data in B, the scores are updated as:

$$S_t(x_i) = \begin{cases} dissim(x_i, x_{i^+}), & x_i \in B'_t, \\ S_{t-1}(x_i), & \text{otherwise.} \end{cases} \tag{4-8}$$

In the above equation, if x_i needs scoring, its score is computed by Eq.(4-2). Otherwise, its score in the last iteration is copied to save computation. By lazying scoring, the computation overhead of contrast scoring is effectively reduced to about $\frac{1}{T}$ of that without lazy scoring, while the accuracy is preserved.

4.4 Experiments

In this section, we first evaluate the *accuracy* with different labeling ratios. Then, we evaluate the *learning speed* of the proposed framework. After that, we evaluate the *reduced computation overhead* by lazy scoring. Finally, we evaluate the impact of *buffer size*.

4.4.1 Experimental Setup

4.4.1.1 Datasets and Evaluation Protocols

We use multiple datasets, including CIFAR-10, CIFAR-100 [56], SVHN [75], and ImageNet-20/50/100 [87] to evaluate the proposed approaches. To perform classification, the encoder is first trained by the proposed approaches to generate data representations. As we mentioned before, training a classifier without any labels does not generate meaningful accuracy. Therefore, we train a classifier with 1%, 10%, or 100% labeled data on the learned encoder.

4.4.1.2 Strength of Temporal Correlation (STC)

We use the metric Strength of Temporal Correlation (STC) to represent the temporal correlation of the input stream. STC represents how many consecutive data in the input stream are from the same class until a class change happens [32]. A larger STC represents a stronger temporal correlation.

4.4.1.3 Default Training Setting

We use ResNet-18 as the base encoder. We train the encoder with the contrastive loss [18] by the Adam optimizer. While the proposed approaches can be applied to both training *from scratch* and *fine-tuning* a pre-trained model, to avoid any bias in the pre-trained model on any approach to compare with, we train *from scratch*. Unless otherwise specified, the batch size is 256 with a weight decay of 0.0001. For subsets of ImageNet, the learning rate is 0.0004, the temperature τ is 0.07, and STC is 100. The model is trained for 300 epochs for ImageNet-20/50 and 100 epochs for ImageNet-100. For CIFAR-10, CIFAR-100, and

SVHN, the learning rate is 0.0001, the temperature τ is 0.5, and the model is trained for 500 epochs with STC 500. For all datasets, the classifier is trained for 500 epochs with Adam optimizer and a learning rate of 0.0003. The lazy scoring is disabled by default to have a fair comparison of different data replacement approaches. The results are averaged over three runs on 2 Nvidia V100 GPUs with different random seeds.

4.4.1.4 Baselines

We first compare the proposed framework with supervised learning using 1% or 10% labeled data. Then, we compare the proposed contrast scoring with four data selection baselines which select data from *unlabeled streaming*. The first two baselines are popular and effective strategies for maintaining exemplars in continual learning while not requiring labels. *Random replacement* is a variant of reservoir sampling [101] and is recently used for continual learning [32]. It selects data uniformly at random from new data and data in the buffer to form the new data buffer. *FIFO replacement* is also recently employed for continual learning [32]. It replaces the oldest data in the buffer with new data. While not requiring labeling information and are seemingly simple, these two approaches have demonstrated superior performance in maintaining data for continual learning compared with approaches that rely on exact labels [17]. The next two baselines are SOTA approaches to select data for efficient training and improving accuracy. *Selective-Backprop* [49] selects data with the largest losses for training. *K-Center* is a SOTA active learning approach [90], which selects the most representative data by performing k-center clustering in the features space. For brevity, in the figures and tables below, we use the term ***Contrast Scoring*** to refer to our proposed approach, and use ***Random Replace***, ***FIFO Replace***, ***Selective-BP***, and ***K-Center*** to refer to the baseline methods.

4.4.2 Improved Accuracy with Different Labeling Ratios

We first compare the proposed framework with supervised learning using 1% or 10% labeled data. The proposed approaches achieve significantly higher accuracy compared to supervised learning. Specifically, the supervised learning approach achieves an accuracy

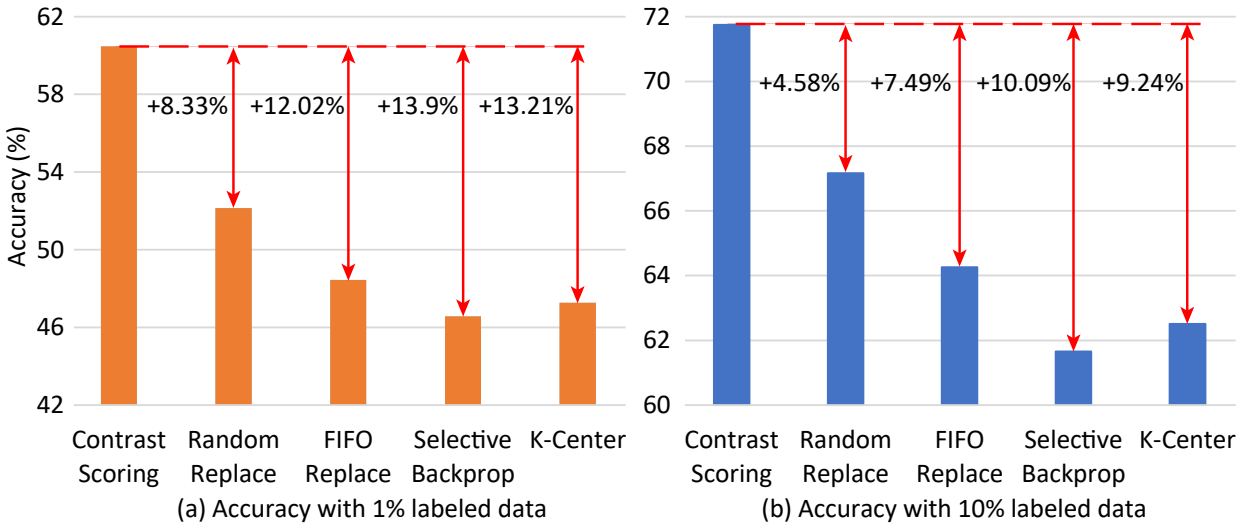


Figure 34: Accuracy on CIFAR-10 with 1% and 10% labeled data.

of 32.11% and 40.53%, which is 28.36% and 31.22% lower than the proposed approaches, respectively. Therefore, supervised learning is not a practical option, and we will focus on evaluating the accuracy of the proposed framework with different data selection approaches.

We compare the proposed contrast scoring with other data selection approaches in terms of accuracy by first performing contrastive learning on unlabeled data with different approaches, and then learning the classifier with different ratios of labeled data (i.e. 1%, 10%).

The proposed data selection approach by contrast scoring substantially outperforms the SOTA baselines. The accuracy with different labeling ratios (i.e. 1%, 10%) on CIFAR-10 is shown in Figure 34, in which contrastive learning is performed for 100 epochs without labels before training the classifier. First, with 1% and 10% labeled data for learning the classifier, the proposed Contrast Scoring approach achieves accuracies of 60.47% and 71.75%, respectively. These accuracies outperform the other four approaches by margins of {8.33%, 12.02%, 13.9%, 13.21%} and {4.58%, 7.49%, 10.09%, 9.24%}, respectively. Second, with fewer labels (i.e. 1% vs. 10%), the proposed contrast scoring outperforms each baseline by a larger margin. This is because with fewer labels, the quality of learned representation becomes more important, and the proposed framework learns better representations than

the baselines. Different from this, the proposed Contrast Scoring selects data that benefit contrastive learning the most.

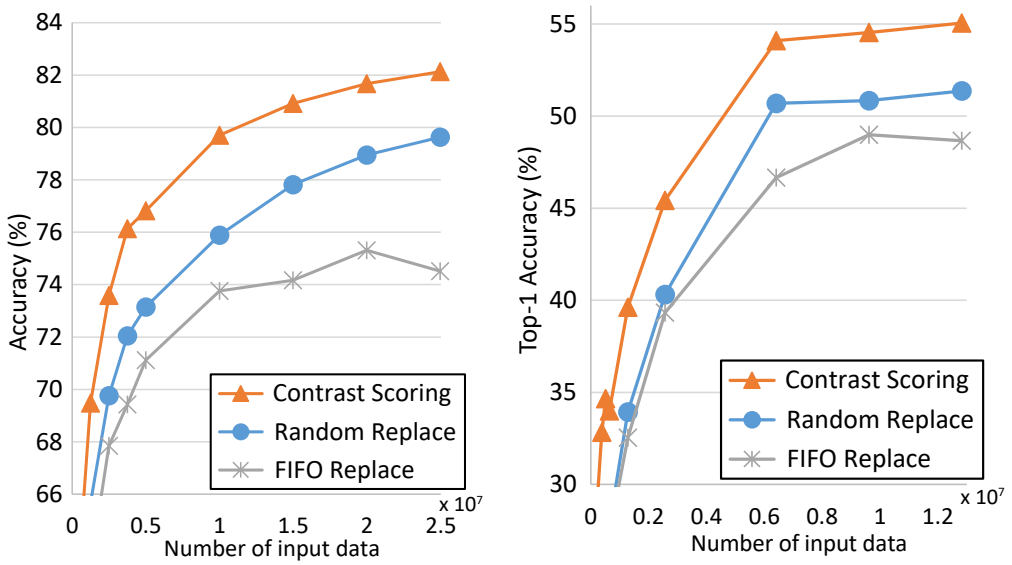
The results show that the most competitive baselines are the two seemingly simple, yet surprisingly effective approaches *Random Replace* and *FIFO Replace*. These results match the results in [14], where a random replacement policy outperforms elaborately designed approaches.

4.4.3 Learning Curve: Improved Learning Speed and Accuracy

We evaluate the learning curve of the proposed approaches and baselines on CIFAR-10, ImageNet-20, ImageNet-50, ImageNet-100, SVHN, and CIFAR-100 datasets. The learning curve represents how fast the model learns representations from the new inputs. Since we aim to evaluate the contrastive learning process by different data selection approaches, to avoid the influence of different label ratios in training the classifier, in the following evaluations, we will use 100% labeled data to train the classifier after contrastive learning and only compare with the two most competitive baselines.

4.4.3.1 Learning Curve on CIFAR-10

The proposed data replacement policy quickly learns data representations and achieves a significantly faster learning speed and a higher accuracy than the baselines. The learning curve on CIFAR-10 is shown in Figure 35 (a). The x-axis is the number of seen inputs and the y-axis is the accuracy. The accuracy of the proposed approaches quickly increases to 76.1% with 3.74M seen data, which is 2.67× faster than the random replacement policy that needs 9.98M data to achieve similar accuracy. The FIFO replacement policy cannot achieve this accuracy even with 25M data. Besides, the proposed approaches achieve much higher final accuracy than the baselines. The proposed approaches achieve a final accuracy of 82.13%, while the random and FIFO replacement policies only achieve 79.63% and 74.51%, respectively.



(a) CIFAR-10.

(b) ImageNet-100.

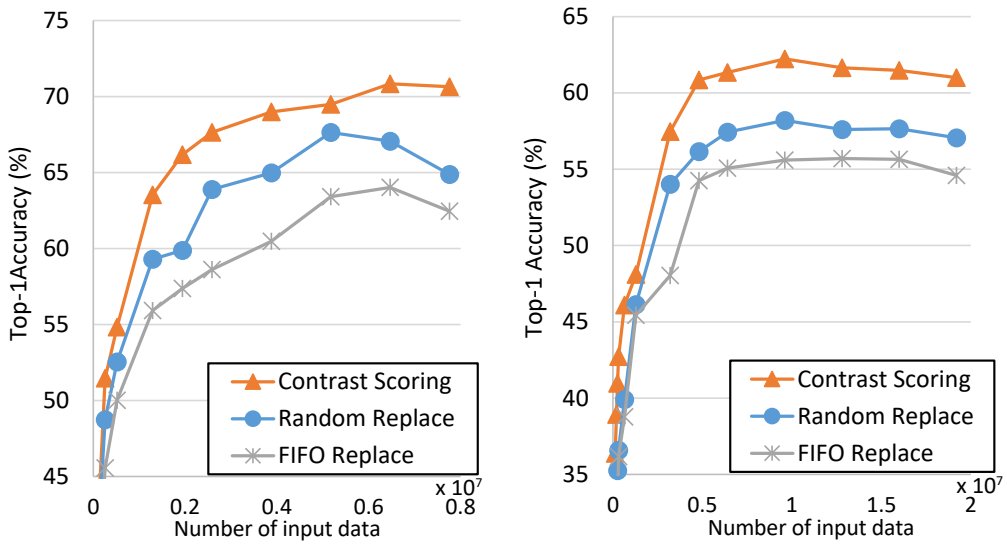
Figure 35: Learning curve on CIFAR-10 and ImageNet-100 datasets.

4.4.3.2 Learning Curve on ImageNet-100

We further evaluate the proposed approaches on the ImageNet-100 dataset. While this dataset is a subset of the large-scale ImageNet dataset, it still features high-resolution images and is challenging for the stream setting. As shown in Figure 35 (b), the proposed approaches achieve a consistently faster learning speed than the baselines. The proposed approaches achieve 55.05% top-1 accuracy and outperform the baselines by 3.69% and 6.39%, respectively.

4.4.3.3 Learning Curve on ImageNet-20 and ImageNet-50

We evaluate the proposed approaches on the ImageNet-20 and ImageNet-50 datasets. As shown in Figure 36, the proposed approaches achieve a significantly faster learning speed and higher accuracy than the baselines. On ImageNet-20, the proposed approaches achieve 70.64% top-1 accuracy and outperform two baselines by 5.76% and 8.19%, respectively. On



(a) ImageNet-20. (b) ImageNet-50.

Figure 36: Learning curve on ImageNet-20 and ImageNet-50 dataset.

ImageNet-50, the proposed approaches achieve 60.99% top-1 accuracy and outperform the baselines by 3.94% and 6.39%, respectively.

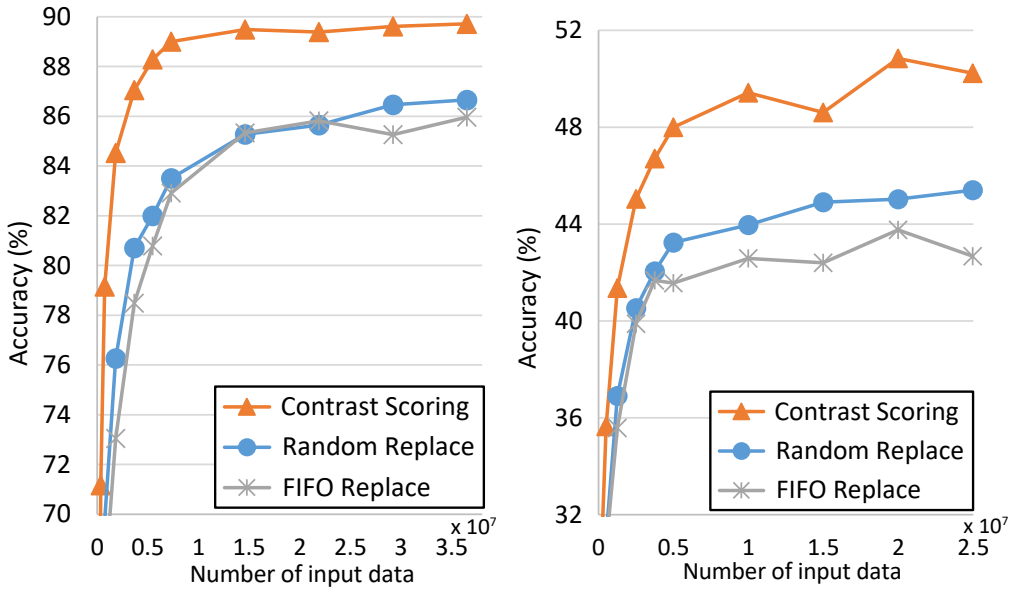
4.4.3.4 Learning Curve on SVHN and CIFAR-100

We evaluate the learning curve on the SVHN and CIFAR-100 datasets, and the results are shown in Figure 37. The proposed approaches substantially outperform the baselines.

4.4.4 The Impacts of Lazy Scoring

We also evaluate the impact of lazy scoring on the accuracy, runtime overhead, and average percent of re-scored data in the buffer in each training iteration. The model is trained on the CIFAR-10 dataset with buffer size 256 and STC 500.

Lazy scoring effectively reduces the additional computation for scoring during training and reduces batch time. As shown in Table 4, when lazy scoring interval T in Eq.(4-7)



(a) SVHN. (b) CIFAR-100.

Figure 37: Learning curve on SVHN and CIFAR-100 datasets.

Table 4: Top-1 accuracy, average re-scoring percent, and batch time (relative to that without scoring) on CIFAR-10 with different lazy scoring intervals.

Lazy Scoring Interval	Disabled	4	20	50	100	200
Accuracy (%)	76.06	77.04 (+0.98)	77.18 (+1.12)	**77.23 (+1.17)**	76.38 (+0.32)	74.22 (-1.84)
Re-scoring Pct. (%)	100.0	21.78	4.31	1.71	0.89	0.44
Relative Batch Time	1.478	1.312	1.232	1.199	1.191	1.172

increases, the average re-scoring percent and the relative batch time (runtime overhead) are effectively reduced. When lazy scoring is not used, each training step of our method is 47.8%

slower than the baselines (without scoring). When lazy scoring is employed with interval 50, each training step is only about 19.9% slower than the baselines. Besides, lazy scoring slightly increases the final accuracy by up to 1.17%. We conjecture that the increased accuracy is because the lazy scoring performs similarly to the momentum encoder in [34]. The score computed multiple iterations ago serves as a momentum score. This slowly updated score brings more information from the past and benefits the data selection.

4.4.5 Improved Accuracy With Different Buffer Sizes

We evaluate the impact of buffer size on the performance of the proposed approaches. The model is trained on the CIFAR-10 dataset. The buffer size is in {8, 32, 128, 256}. The corresponding learning rate is scaled to $\{1, 3, 5, 10\}\times 10^{-5}$, roughly following a learning rate $\propto \sqrt{\text{batch size}}$ scaling scheme.

Table 5: Accuracy on CIFAR-10 dataset with different buffer sizes.

Buffer Size	Method	Accuracy
8	Contrast Scoring (ours)	69.38
	Random Replace	66.71 (-2.67)
	FIFO Replace	65.91 (-3.47)
32	Contrast Scoring (ours)	73.26
	Random Replace	70.65 (-2.61)
	FIFO Replace	70.80(-2.46)
128	Contrast Scoring (ours)	73.97
	Random Replace	71.28 (-2.69)
	FIFO Replace	70.65 (-3.32)
256	Contrast Scoring (ours)	**76.06**
	Random Replace	72.75(**-3.31**)
	FIFO Replace	70.53 (**-5.53**)

The proposed approaches consistently outperform the baselines under different buffer sizes.

As shown in Table 5, under different buffer sizes, the accuracy of the proposed approaches maintains a clear margin over the baselines. Besides, the margin becomes larger as the buffer size increases. This is because a larger buffer size provides the framework a better opportunity to select more informative data, and the proposed approaches can leverage this opportunity to maintain more representative data in the buffer for learning, while the baselines cannot. Also, all the approaches achieve higher accuracy when the buffer size becomes larger. This is because a larger buffer size provides a larger batch size, and contrastive learning naturally benefits from a large batch size since it provides more negative samples [18].

4.5 Video Demo

We have created a video demo showcasing the workflow of the proposed self-supervised on-device learning framework, and the video is available online at [6].

4.5.1 System Setup

The system setup is shown in Figure 38. The system consists of the image source and the mobile GPU platform. The image source is a computer connected to a display (the display on the right-hand side of Figure 38) that shows the images from the CIFAR-10 dataset in random orders [57]. The mobile GPU platform is the Nvidia Jetson TX2 [78]. It has a camera facing the display on the right-hand side to capture the images from the image source. The output from the mobile GPU platform is displayed on a separate screen, which is shown on the left-hand side of Figure 38.

4.5.2 Description of Video Demo

The video demo consists of two stages, as shown in Figure 32. In the first stage, the proposed data selection method is used to perform self-supervised contrastive learning with the unlabeled input stream. The data buffer for contrastive learning can store up to 64 images, and data replacement based on the proposed contrast score is performed after 64

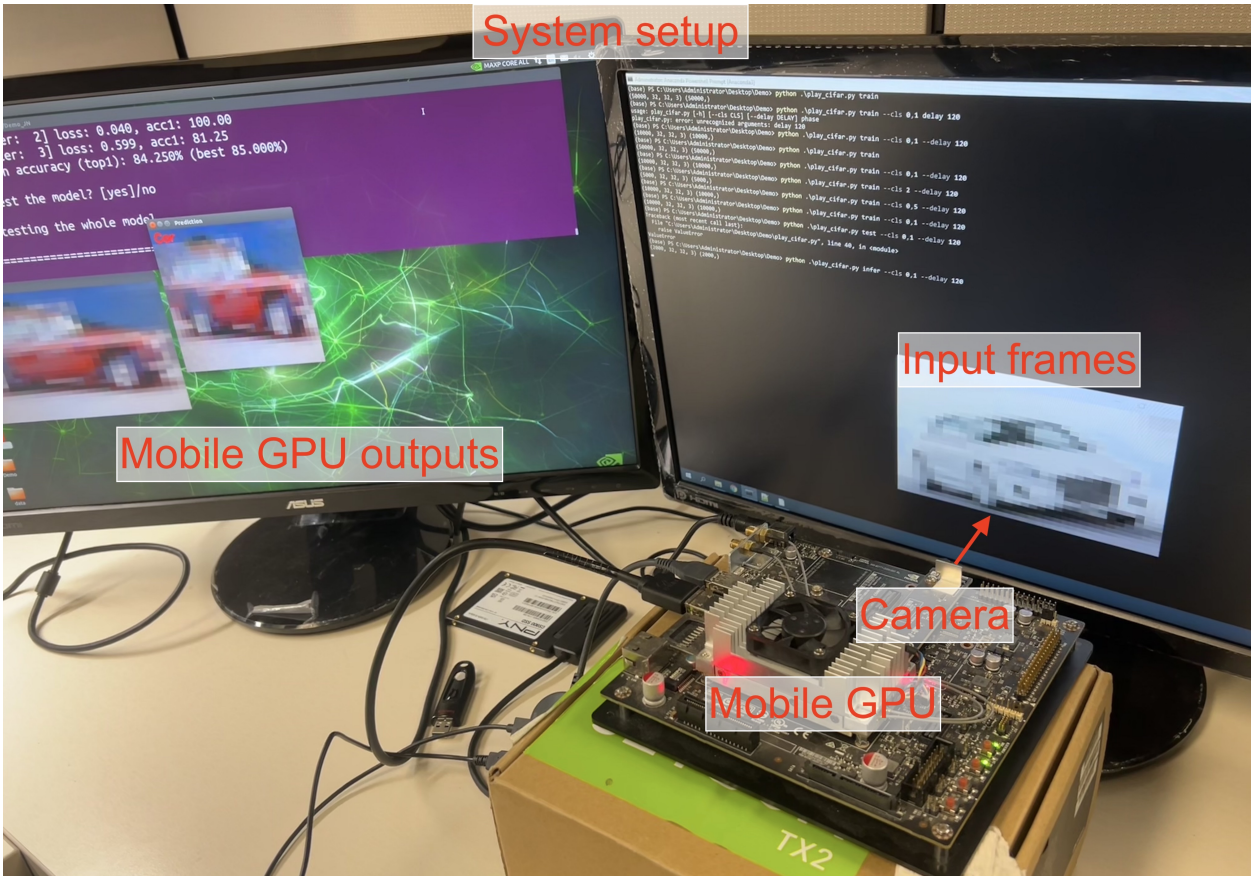


Figure 38: This is the system setup for the video demo, which showcases the self-supervised on-device learning framework.

images have been captured by the camera. The selected data is then used to update the encoder. In the second stage, a classifier is learned using a few labeled data on top of the learned representations. In this demo, 64 images are manually labeled and used to train the classifier. Finally, the trained encoder and classifier make predictions for each new image.

4.6 Summary

This work aims to enable on-device contrastive learning from input streaming data. We propose a framework to maintain a small data buffer filled with the most representative data for learning. To achieve the data selection without requiring labels, we propose a data replacement policy by contrast scoring. To reduce the runtime overhead of data scoring, we propose a lazy scoring strategy. Experimental results on multiple datasets show that the proposed approaches achieve superior learning speed and accuracy compared with SOTA baselines.

5.0 Collaborative Unsupervised Learning with Distributed Devices

This chapter presents a project that enables distributed edge devices for learning a shared model from decentralized unlabeled data by feature fusion and neighborhood matching [110]. It is organized as follows. First, motivation is introduced. Next, the background and the related works are presented. Then, the details of the proposed collaborative unsupervised learning techniques to learn representations from decentralized unlabeled data are presented. After that, the experimental results are shown to demonstrate the effectiveness of the proposed techniques for representation learning from decentralized devices. Finally, the conclusion is presented to summarize this project.

5.1 Introduction

Collaborative learning is an effective method for multiple distributed clients to jointly learn a shared model from decentralized data. This approach enables clients to leverage their individual data to generate a stronger, more comprehensive model. In the learning process, each client updates the local model by using local data, and then a central server aggregates the local models to obtain a shared model. In this way, collaborative learning enables learning from decentralized data [73] while keeping data local for privacy. Decentralized collaborative learning has the potential to revolutionize healthcare applications, particularly in the early detection of cognitive diseases such as Parkinson's and assessment of mental health. This approach involves multiple personal devices, such as mobile phones, collaborating to learn and provide timely warnings [19]. Collaborative learning can also be used for robotics, in which multiple robots learn a shared navigation scheme to adapt to new environments [66]. Compared with local learning, collaborative learning improves navigation accuracy by utilizing knowledge from other robots.

Existing collaborative learning approaches assume local data is fully labeled so that supervised learning can be used for the model update on each client. However, labeling all the

data is usually unrealistic due to high labor costs and the requirement for expert knowledge. For example, in medical diagnosis, even if the patients are willing to spend time on labeling all the local data, the deficiency of expert knowledge of these patients will result in large label noise and thus an inaccurate learned model. The deficiency of labels makes supervised collaborative learning impractical. Self-supervised learning can address this challenge by pre-training a neural network encoder with unlabeled data, followed by fine-tuning for a downstream task with limited labels. Contrastive learning (CL), an effective self-supervised learning approach [18], can learn data representations from unlabeled data to improve the model. By integrating CL into collaborative learning, clients can collaboratively learn data representations by using a large amount of data without labeling.

In collaborative learning, data collected on clients are inherently far from IID [39], which results in two unique challenges when integrating collaborative learning with CL as collaborative contrastive learning (CCL) to learn high-quality representations. The *first challenge* is that each client only has a small amount of unlabeled data with limited diversity, which prevents effective contrastive learning. More specifically, compared with the global data (the concatenation of local data from all clients), each client only has a subset of the global data with a limited number of classes [73, 113, 124]. For instance, in real-world datasets [69], each client only has one or two classes out of seven object classes. Since conventional contrastive learning frameworks [18, 34] are designed for centralized learning on large-scale datasets with sufficient data diversity, directly applying them to local learning on each client will result in the low quality of learned representations.

The *second challenge* is that each client focuses on learning its local data without considering the data of the other clients. As a result, the features of data in the same class but from different clients may not be well-clustered even though they could have been clustered for improved representations. Data are decentralized in collaborative learning and even if two clients have data of the same class, they are unaware of this fact and cannot leverage it to collaboratively learn to cluster these data. Furthermore, in scenarios where no labels are available, identifying the correct data clusters and performing clustering for improved representations can be a challenging task even if one client has knowledge of another's data.

To address these challenges, we propose a collaborative contrastive learning (CCL) framework to learn visual representations from decentralized unlabeled data on distributed clients. The framework employs contrastive learning [34] for local learning on each client and consists of two approaches to learning high-quality representations. The first approach is feature fusion, which provides remote features as accurate contrastive information to each client for better local learning. To protect the privacy of remote features against malicious clients, we employ an encryption method [42] to encrypt the images before generating their features. The second approach is neighborhood matching and it further aligns each client's local features to the fused features such that well-clustered features among clients are learned.

In summary, the main contributions of the paper include:

- **Collaborative contrastive learning framework.** We propose a framework with two approaches to learning visual representations from unlabeled data on distributed clients. The first approach improves the local representation learning on each client with limited data diversity, and the second approach further learns unified global representations among clients.
- **Feature fusion for better local representations.** We propose a feature fusion approach to leverage remote features for better local learning while avoiding raw data sharing. The remote features serve as negatives in the local contrastive loss to achieve a more accurate contrast with fewer false negatives and more diverse negatives.
- **Neighborhood matching for improved global representations.** We propose a neighborhood matching approach to cluster decentralized data across clients. During local learning, each client identifies the remote features to cluster local data with and performs clustering. In this way, well-clustered features among clients can be learned.

5.2 Background and Related Work

5.2.1 Contrastive Learning

Contrastive learning is a self-supervised approach to learning an encoder (i.e. a convolutional neural network without the final classifier) for extracting visual representation vectors from the unlabeled input images by performing a proxy task of instance discrimination [18, 34, 111, 117]. In this work, we employ the contrastive learning framework from [34]. For an input image x, its representation vector z is obtained by $z = f(x),\ z \in \mathbb{R}^d$, where $f(\cdot)$ is the encoder. Let the representation vectors *query* q and *key* k^+ form a positive pair, which are the representation vectors from two transformations (e.g. cropping and flipping) of the same input image. Let Q be the *memory bank* with K representation vectors stored, serving as negatives. The positive pair *query* q and *key* k^+ will be contrasted with each vector $n \in Q$ (i.e. negatives) by the loss function:

$$\ell_q = -\log \frac{\exp(q \cdot k^+/\tau)}{\exp(q \cdot k^+/\tau) + \sum_{n \in Q} \exp(q \cdot n/\tau)}. \tag{5-1}$$

Minimizing this loss will learn an encoder to generate visual representations. Then a classifier can be trained on top of the encoder by using limited labeled data.

However, existing contrastive learning approaches are designed for centralized learning [18, 34] and require sufficient data diversity for learning. When applied to each client in collaborative learning with limited data diversity, their performance will greatly degrade. Therefore, an approach to increase the local data diversity of each client while protecting the shared information is needed.

5.2.2 Collaborative Learning

The goal of collaborative learning is to learn a shared model by aggregating locally updated models from clients while keeping raw data on local clients [73]. In collaborative learning, there are C clients indexed by c. The training data D is distributed among clients, and each client c has a subset of the training data $D_c \subset D$. There are recent works aiming to optimize the aggregation process [76, 84]. While our work can be combined with these

works, for simplicity, we employ a typical collaborative learning algorithm [73]. The learning is performed round-by-round. In communication round t, the server randomly selects $\beta \cdot C$ clients C^t and sends them the global model with parameters θ^t, where β is the percentage of active clients per round. Each client $c \in C^t$ updates the local parameters θ_c^t on local dataset D_c for E epochs to get θ_c^{t+1} by minimizing the loss $\ell_c(D_c, \theta^t)$. Then the local models are aggregated into the global model by averaging the weights $\theta^{t+1} \leftarrow \sum_{c \in C^t} \frac{|D_c|}{\sum_{i \in C^t} |D_i|} \theta_c^{t+1}$. This learning process continues until the global model converges.

To improve the performance of collaborative learning on non-IID data, [46, 124] share local raw data (e.g. images) among clients. However, sharing raw data among clients will cause privacy concerns [64]. Besides, they need fully labeled data to perform collaborative learning, which requires expert knowledge and potentially high labeling costs. Therefore, an approach to performing collaborative learning with limited labels and avoiding sharing raw data is needed.

5.3 Collaborative Unsupervised On-device Learning

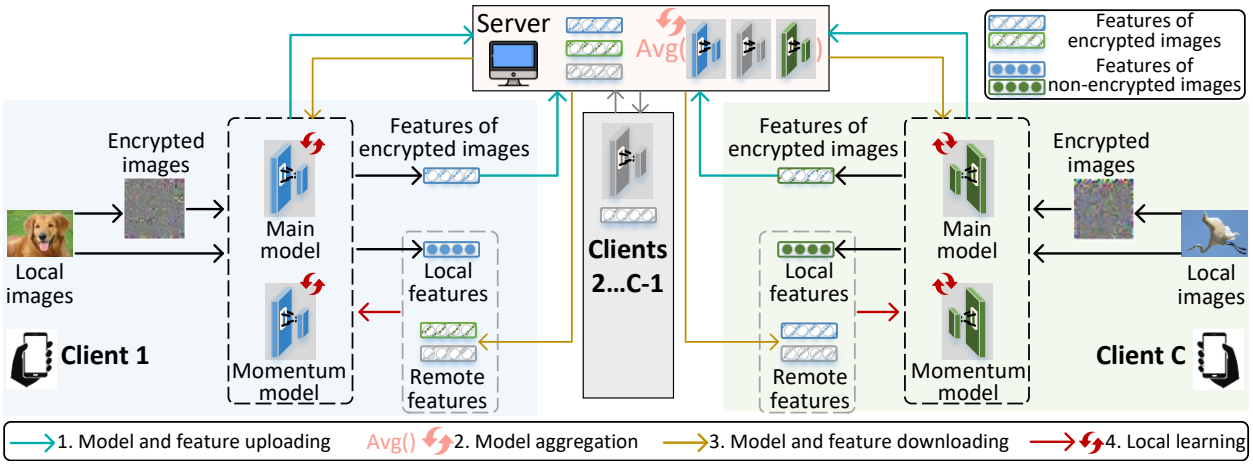


Figure 39: Overview of the proposed collaborative contrastive learning (CCL) framework.

We propose a collaborative contrastive learning (CCL) framework to learn representations from unlabeled data on distributed clients. These distributed data cannot be combined in a

single location to construct a centralized dataset due to privacy and legal constraints [52]. The overview of the proposed framework is shown in Figure 39. There is a central server that coordinates multiple clients to learn representations. The local model for each client is based on MoCo [34]. CCL follows the proposed feature fusion technique to reduce the false negative ratio on each client for better local learning (Section 5.4). Besides, based on fused features, CCL further uses the proposed neighborhood matching to cluster representations of data from different clients to learn unified representations among clients (Section 5.5).

Before introducing the details of feature fusion and neighborhood matching, we present the proposed CCL process. CCL is performed round-by-round and there are four steps in each round as shown in Figure 39. First, each client c uploads its latest model (consisting of the main model f_q^c and momentum model f_k^c) and latest features $\overline{Q}_{l,c}$ of encrypted images to the server. Second, the server aggregates main models from clients by $\theta_q \leftarrow \sum_{c \in C} \frac{|D_c|}{|D|} \theta_q^c$ and momentum models by $\theta_k \leftarrow \sum_{c \in C} \frac{|D_c|}{|D|} \theta_k^c$ to get updated f_q and f_k, where $|D_c|$ is the data size of client c and $|D|$ is the total data size of $|C|$ clients. The server also combines features as $\overline{Q} = \{\overline{Q}_{l,c}\}_{c \in C}$. Third, the server downloads the aggregated models f_q and f_k and combined features $\overline{Q}$ excluding $\overline{Q}_{l,c}$ as $Q_{s,c} = \{\overline{Q} \setminus \overline{Q}_c\}$ to each client c. Fourth, each client updates its local models with f_q and f_k, and then performs local contrastive learning for multiple epochs with local features $Q_{l,c}$ and remote features $Q_{s,c}$ by using loss Eq.(5-13) including contrastive loss with fused features Eq.(5-6) and neighborhood matching Eq.(5-12). During local contrastive learning, to generate features $\overline{Q}_{l,c}$ for uploading in the next round, images x are encrypted by *InstaHide* [42] as $\tilde{x}$ and fed into momentum model f_k^c. In this way, even if a malicious client can ideally recover $\tilde{x}$ from the features $\overline{Q}_{l,c}$, which is already very unlikely in practice, it still cannot reconstruct x from $\tilde{x}$ since *Instahide* effectively hides information contained in x. Next, we present the details of local contrastive learning, including feature fusion to reduce false negative ratio in Section 5.4 and neighborhood matching for unified representations in Section 5.5.

5.4 Local Learning with Feature Fusion

Next, we focus on how to perform local CL in each round of CCL. We first present the key challenge of CL on each client, which does not exist in conventional centralized CL. Then we propose feature fusion to tackle this challenge and introduce how to perform local CL with fused features.

5.4.1 Key Challenge

Limited data diversity causes a high false negative (FN) ratio for each client. A low FN ratio is crucial to achieving accurate CL [53]. For one image sample q, FNs are features that we use as negative features but actually correspond to images of the same class as q. In centralized CL, the percentage of FNs is inherently low since diverse data are available. The model has access to the whole dataset D with data from all the classes instead of a subset D_c as in collaborative learning. Thus, when we randomly sample negatives from D, the FN ratio is low. For instance, when dataset D has 1000 balanced classes and the negatives n are randomly sampled, for any image q to be learned, only $\frac{1}{1000}$ of n are from the same class as q and are FNs.

However, in collaborative learning, the FN ratio is inherently high for each client due to the limited data diversity, which significantly degrades the performance of contrastive learning. For instance, in real-world datasets [69], one client can have only one or two classes out of seven classes. With limited data diversity on each client, when learning image sample q, many negatives n to contrast with will be from the same class as q and are FNs. To perform contrastive learning by minimizing the contrastive loss in Eq.(5-1), the model scatters the FNs away from q, which should have been clustered since they are from the same class. As a result, the representations of samples from the same class will be scattered instead of clustered and degrade the learned representations.

5.4.2 Feature Fusion

To address this challenge, we propose feature fusion to share negatives in the feature space (i.e. the output vector of the encoder), which reduces FN and improves the diversity of negatives while avoiding raw data sharing. Let $Q_{l,c}$ be the memory bank of size K for local features of non-encrypted images on client c, and let $\overline{Q}_{l,c}$ be features of encrypted images. In one round t of CCL, features $\overline{Q}_{l,c}$ of encrypted images on each client c will be uploaded to the server (i.e. step 1 in Figure 39). The server also downloads combined features $\overline{Q}$ excluding $\overline{Q}_{l,c}$ to each client c (i.e. step 3 in Figure 39) to form its memory bank of remote negatives $Q_{s,c}$ as follows.

$$Q_{s,c} = \{\overline{Q}_{l,i} \mid 1 \leq i \leq |C|, i \neq c\}, \tag{5-2}$$

where C is the set of all clients.

On client c, with local negatives $Q_{l,c}$ and remote negatives $Q_{s,c}$, the loss for sample q is defined as:

$$\ell_q = -\log\left[\frac{\exp(q \cdot k^+/\tau)}{\exp(q \cdot k^+/\tau) + \sum_{n \in \{Q_l \cup Q_s\}} \exp(q \cdot n/\tau)}\right], \tag{5-3}$$

where we leave out the client index c in $Q_{l,c}$ and $Q_{s,c}$ for conciseness. ℓ_q is the negative log-likelihood over the probability distribution generated by applying a softmax function to a pair of input q and its positive k^+, negatives n from both local negatives Q_l and remote negatives Q_s.

5.4.2.1 Effectiveness of Feature Fusion

The remote negatives Q_s reduce the FN ratio in local contrastive learning and improve the quality of learned representations on each client. More specifically, in collaborative learning with non-IID data, we assume the global dataset D has M classes of data, each class with the same number of data. Each client $c \in C$ has a subset $D_c \subset D$ of the same length in m classes ($m \leq M$) [73, 124]. For a sample q on client c, when only local negatives $Q_{l,c}$ are used, $\frac{1}{m}|Q_{l,c}|$ negatives will be in the same class as q, which results in an FN ratio $R_{FN} = \frac{1}{m}$. Since m is usually small (e.g. 2) due to limited data diversity, the FN ratio R_{FN} will be large (e.g. 50%) and degrade the quality of learned representations. Different from this, when remote

negatives are used, the FN ratio is:

$$R_{FN}(q) = \frac{\frac{1}{m}|Q_{l,c}| + \sum_{i \in C, i \neq c} \mathbb{I}(i,q)\frac{1}{m}|Q_{l,i}|}{|Q_{l,c}| + \sum_{i \in C, i \neq c} |Q_{l,i}|} \leq \frac{1}{m}, \tag{5-4}$$

where $\mathbb{I}(i,q)$ is an indicator function that equals 1 when client i has data of the same class as q, and 0 otherwise.

In most cases, $R_{FN}(q)$ is effectively reduced by the remote negatives. *First*, in the extreme case of non-IID data distribution, where the classes on each client are mutually exclusive [124], all $\mathbb{I}(i,q)$ equal 0 and $R_{FN}(q) = \frac{1}{m}\frac{|Q_{l,c}|}{|Q_{l,c}|+\sum_{i \in C, i \neq c}|Q_{l,i}|} = \frac{1}{m|C|} \ll \frac{1}{m}$. With the remote negatives, the FN ratio is effectively reduced by a factor $|C|$. *Second*, as long as not all clients have data of the same class as q, some elements in $\{\mathbb{I}(i,q)\}_{i=1,\ i \neq c}^{|C|}$ will be 0, and $R_{FN}(q)$ in Eq.(5-4) will be smaller than $\frac{1}{m}$. In this case, the FP ratio is also reduced. *Third*, even if the data on each client is IID and all $\mathbb{I}(i,q)$ equal 1, which is unlikely in realistic collaborative learning [39], the FN ratio $R_{FN}(q)$ will be $\frac{1}{m}$. In this case, while $R_{FN}(q)$ is the same as that without remote negatives, the increased diversity of negatives from other clients can still benefit local contrastive learning.

5.4.2.2 Further Reducing the False Negative Ratio

To further reduce the FN ratio, we propose to exclude the local negatives by removing Q_l in the denominator of Eq.(5-3) and only keeping remote negatives Q_s. The corresponding FN ratio becomes:

$$R'_{FN}(q) = \frac{\sum_{i \in C, i \neq c} \mathbb{I}(i,q)\frac{1}{m}|Q_{l,i}|}{\sum_{i \in C, i \neq c} |Q_{l,i}|} \leq R_{FN}(q). \tag{5-5}$$

When not all other clients have data in the same class as q, some $\mathbb{I}(i,q)$ values will be 0. As a result, $R'_{FN}(q) < R_{FN}(q)$, which leads to a further reduction in the FN ratio. Based on the loss ℓ_q for one sample q in Eq.(5-3), the contrastive loss for one mini-batch B is:

$$\mathcal{L}_{contrast} = \frac{1}{|B|}\sum_{q \in B} \ell_q. \tag{5-6}$$

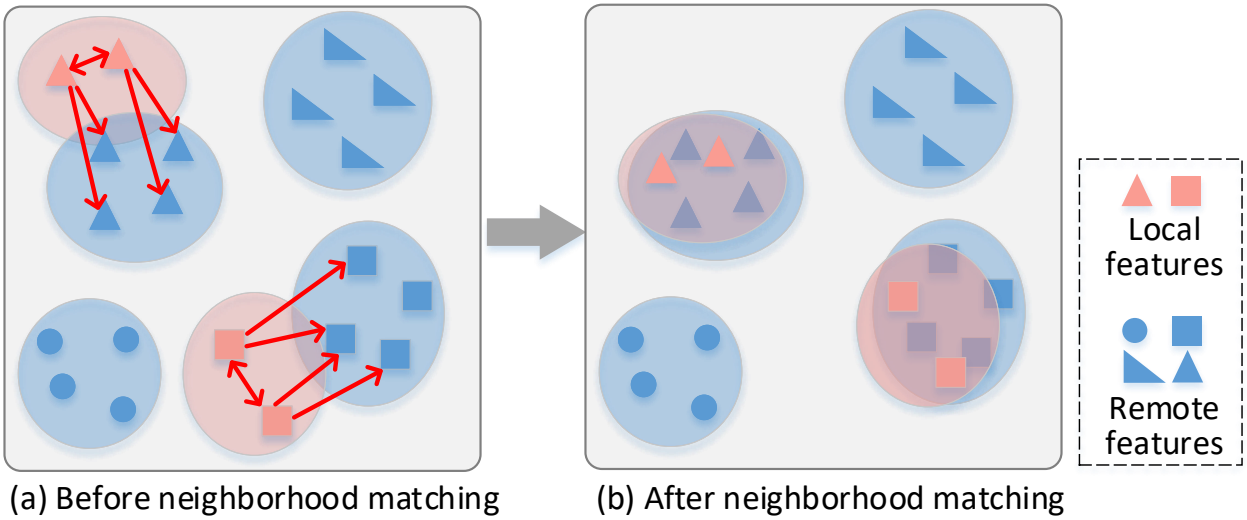


Figure 40: Neighborhood matching aligns each client's local features to the remote features such that well-clustered features among clients are learned.

In local contrastive learning, each client focuses on learning its local data without considering data on the other clients. As a result, the features of data in the same class but from different clients may not be well-clustered even though they could be clustered for improved representations.

5.5.1 Challenge

To cluster local features correctly with remote features, it is necessary to identify local data and remote features belonging to the same class. However, this is challenging in the absence of labels for local data and remote features, making it difficult to identify the correct clusters for local features.

To tackle this challenge, we propose a neighborhood matching approach that identifies the remote features to cluster local data with. We also define an objective function to perform the clustering. First, during local learning on one client, as shown in Figure 40, for each local sample we find N nearest features from both the *remote* and *local* features

as neighbors. Then the features of the local sample will be pushed to these neighbors by the proposed entropy-based loss. Since the model is synchronized from the server to clients in each communication round, the remote and local features are encoded by similar models on different clients. Therefore, the neighbors are likely to be in the same class as the local sample being learned, and clustering them will improve the learned representations of global data. In this way, the global model is also improved when aggregating local models.

5.5.2 Identifying Neighbors

To push each local sample close to its neighbors, we minimize the entropy of one sample's matching probability distribution to either a remote feature or a local feature. To improve the robustness, we match one sample to N nearest features at the same time, instead of only one nearest feature. By minimizing the entropy for N nearest neighbors, the sample's matching probability to each of the nearest neighbors will be individually certain.

For each local sample q_i, we regard top-N closest features, either from local or remote features, as neighbors. To find the neighbors, we first define the neighbor candidates as:

$$Q' = \{Q_{s+l,i} |\ i \sim \mathcal{U}(|Q_s| + K, K)\}\,, \tag{5-7}$$

where $i \sim \mathcal{U}(|Q_s| + K, K)$ samples K integer indices from $[|Q_s| + K]$ randomly at uniform. $Q_{s+l,i}$ is the element with index i in the union of remote and local features $Q_s \cup Q_l$. For one local sample q_i, the neighbors $P(q_i)$, which are the top-N nearest neighbor candidates Q', is given by:

$$P(q_i) = \{Q'_j \mid j \in topN(S_i)\}, \tag{5-8}$$

$$S_i = \{S_{i,m} \mid 1 \leq m \leq K\} \tag{5-9}$$

where $S_{i,m} = sim(q_i, n_m) = q_i^T \cdot n_m / \|q_i\| \|n_m\|$ is the cosine similarity between q_i and one neighbor candidate $n_m \in Q'$.

5.5.3 Neighborhood Matching Loss

To make the probability of q_i matching to each $n_j \in P(q_i)$ individually certain, we consider the set:

$$L_j = \{n_j\} \cup \{Q' \setminus P(q_i)\} \in \mathbb{R}^{(K-N+1)\times d}, \tag{5-10}$$

where d is the dimension of one feature vector. L_j contains one of the top-N nearest neighbors n_j and neighbor candidates excluding all other top-N nearest neighbors.

Given L_j, the probability that sample q_i is matched to one of the neighbors $n_a \in L_j$ is:

$$p_{i,j,a} = \frac{\exp(q_i^T \cdot n_a/\tau_{nm})}{\sum_{n\in L_j} \exp(q_i^T \cdot n/\tau_{nm})},\ n_a \in L_j. \tag{5-11}$$

The temperature τ_{nm} controls the softness of the probability distribution [38]. Since n_j has the largest cosine similarity with q_i for $n \in L_j$, $p_{i,j,a}$ will have the largest value when $n_a = n_j$ for $n_a \in L_j$. In this way, when minimizing the entropy of the probability distribution $\{p_{i,j,a}\}_{n_a\in L_j}$, the matching probability of q_i and n_j will be maximized.

For one mini-batch B, to match each sample to its N nearest neighbors, the entropy for all samples in this mini-batch is calculated as:

$$\mathcal{L}_{neigh} = -\frac{1}{|B|}\sum_{i\in B}\frac{1}{N}\sum_{j=1}^{N}\sum_{a=1}^{K-N+1} p_{i,j,a}\log(p_{i,j,a}), \tag{5-12}$$

where $K-N+1$ is the number of features in L_j, and N is the number of nearest neighbors to match. By minimizing $\mathcal{L}_{neigh}$, each $i \in B$ will be aligned to its top-N nearest neighbors.

5.5.4 Final Loss

Based on the contrastive loss with fused features in Eq.(5-6) and neighborhood matching loss in Eq.(5-12), the overall objective is formulated as:

$$\mathcal{L} = \mathcal{L}_{contrast} + \lambda\mathcal{L}_{neigh}, \tag{5-13}$$

where λ is a weight parameter.

5.6 Experiments

5.6.1 Datasets and Model Architecture

We evaluate the proposed approaches on three datasets, including CIFAR-10 [56], CIFAR-100 [56], and Fashion-MNIST [118]. Both CIFAR-10 and CIFAR-100 have 10 classes with 50k samples for training and 10k samples for testing, and Fashion-MNIST has 10 classes with 60k samples for training and 10k samples for testing. We use ResNet-18 as the base encoder and use a 2-layer MLP to project the representations to 128-dimensional feature space [18, 34]. For the model ResNet-18, following [18], we replace the first 7×7 Conv of stride 2 with 3×3 Conv of stride 1, and remove the first pooling layer.

5.6.2 Distributed Setting of Collaborative Contrastive Learning

We evaluate the proposed approaches in three collaborative learning settings for each of the three datasets. The first setting (IID) follows [124] and there are 10 clients. Each client is randomly assigned a uniform distribution over 10 classes for CIFAR-10/FMNIST, and 100 classes for CIFAR-100. Following [73], we set the percentage of active clients per round $\beta = 0.2$ and the number of local epochs $E = 1$. The second setting (non-IID 1) is similar to the first one except that the data distribution is non-IID, where the samples are split to clients by classes, and each client has samples of 2 classes for CIFAR-10/FMNIST and 20 classes for CIFAR-100 [122, 124]. The third setting (non-IID 2) follows [122], where there are 5 clients and each client has samples of 2 classes for CIFAR-10/FMNIST and 20 classes for CIFAR-100, with $\beta = 1.0$ and $E = 5$. For the first two settings, we train the models for $300/\beta$ rounds for CIFAR-10/100 and $100/\beta$ rounds for FMNIST, such that the total number of mini-batches used on all clients are identical to centralized training with 300 epochs and 100 epochs, respectively. For the third setting, following [122], we train the model for 100 rounds on all three datasets.

5.6.3 Training Details of Collaborative Finetuning for Evaluation

The encoder learned by collaborative contrastive learning is used as the initialization for collaborative finetuning, where clients collaboratively finetune the encoder and classifier by supervised collaborative learning with few labeled data (results shown in Section 5.6.9). The distributed setting in collaborative finetuning is the same as collaborative contrastive learning described in Section 5.6.2. For all three collaborative learning settings, we train for $300/\beta$ rounds for CIFAR-10, CIFAR-100, and FMNIST, where β is the percentage of active clients per round described in Section 5.6.2. We use SGD as the optimizer with a batch size of 128 and an initial learning rate of 0.1 with the cosine decay schedule. Standard data augmentations including random cropping and random flipping (probability 0.5) are used.

5.6.4 Metrics

To evaluate the quality of learned representations, we use standard metrics for centralized self-supervised learning, including the *linear evaluation* and *semi-supervised learning* [18]. Besides, we evaluate by *collaborative finetuning* for realistic collaborative learning. In linear evaluation, a linear classifier is trained on top of the frozen base encoder, and the test accuracy represents the quality of learned representations. We first perform collaborative learning by the proposed approaches without labels to learn representation. Then we *fix* the encoder and train a linear classifier on the 100% labeled dataset on top of the encoder. The classifier is trained for 100 epochs by the SGD optimizer following the hyper-parameters from [34]. In semi-supervised learning, we first train the base encoder without labels in collaborative learning. Then we append a linear classifier to the encoder and *finetune* the whole model on 10% or 1% labeled data for 20 epochs with SGD optimizer following the hyper-parameters from [16]. In collaborative finetuning, the learned encoder by the proposed approaches is used as the initialization for finetuning the whole model by supervised collaborative learning [73] with few locally labeled data on clients. Detailed collaborative finetuning settings can be found in the Appendix.

5.6.5 Baselines

We compare the proposed methods with multiple approaches. *Predicting Rotation* is a self-supervised learning approach by predicting the rotation of images [26]. *DeepCluster-v2* is the improved version of DeepCluster [15, 16] and achieves SOTA performance. *SwAV* and *SimCLR* are SOTA approaches for self-supervised learning [16, 18]. We combine these approaches with FedAvg as *FedRot*, *FedDC*, *FedSwAV*, and *FedSimCLR*. *FedCA* is the SOTA collaborative unsupervised learning approach with a shared dictionary and online knowledge distillation [122]. Besides, we compare with two methods as upper bounds. *MoCo* [34] is a centralized contrastive learning method assuming all data are combined in a single location. We compare with MoCo since the local model in the proposed methods is based on it. *FedAvg* [73] is a fully supervised collaborative learning method.

5.6.6 Linear Evaluation

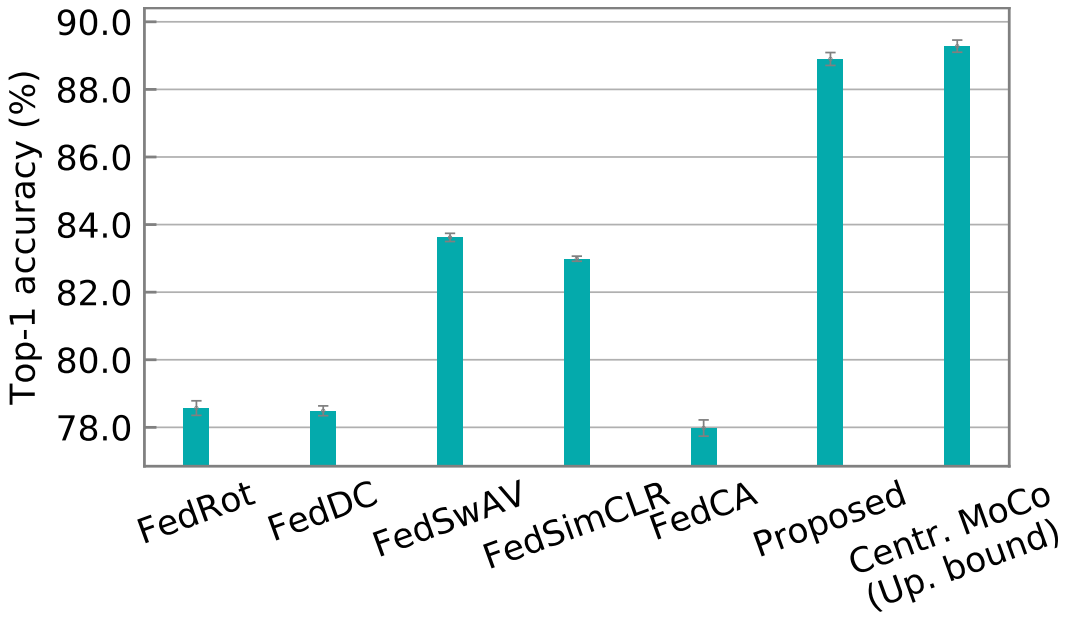


Figure 41: Linear evaluation accuracy on CIFAR-10 in the IID setting.

5.6.6.1 Linear Evaluation on CIFAR-10

We evaluate the proposed approaches by linear evaluation with 100% data labeled for training the classifier on top of the *fixed* encoder learned with unlabeled data by different approaches. The proposed approaches significantly outperform other methods and even match the performance of the centralized upper bound method. The results on CIFAR-10 in the IID setting are shown in Figure 41. On CIFAR-10, the proposed approaches achieve 88.90% top-1 accuracy, only 0.38% below the upper bound method centralized MoCo. The proposed approaches also outperform the SOTA method FedCA by +10.92% top-1 accuracy and the best-performing baseline FedSwAV by +5.28%.

Table 6: Linear evaluation on CIFAR-10, CIFAR-100, and FMNIST datasets under the IID and two non-IID settings.

	CIFAR-10			CIFAR-100			FMNIST		
Method	IID	Non-1	Non-2	IID	Non-1	Non-2	IID	Non-1	Non-2
FedRot	78.57	<u>75.88</u>	70.98	45.80	44.57	43.15	83.74	82.65	82.90
FedDC	78.49	69.97	69.34	49.27	49.06	47.21	88.41	85.92	88.35
FedSwAV	<u>83.62</u>	75.07	75.36	<u>55.51</u>	<u>51.45</u>	<u>53.77</u>	<u>89.63</u>	<u>87.11</u>	<u>89.75</u>
FedSimCLR	82.99	71.23	73.30	48.83	45.67	48.46	88.45	84.41	86.23
FedCA	77.98	75.57	<u>75.50</u>	48.93	47.70	48.22	86.98	86.22	86.46
Proposed	**88.90**	**79.07**	**78.31**	**61.91**	**57.54**	**58.63**	**91.26**	**88.13**	**90.08**
Upper bounds									
MoCo (Centralized)	89.28	—	—	63.72	—	—	91.97	—	—
FedAvg (Supervised)	92.88	60.60	59.03	73.08	67.59	66.90	94.12	77.08	69.92

5.6.6.2 Linear Evaluation on Various Datasets and Distributed Settings

We evaluate the proposed approaches on different datasets and collaborative learning settings. Table 6 presents the results for CIFAR-100 and FMNIST datasets under IID setting

and non-IID settings 1 and 2. The linear classifier is trained on top of the *fixed* encoder learned with unlabeled data by different approaches. Under all three collaborative learning settings and on both datasets, the proposed approaches significantly outperform the baselines.

For example, on CIFAR-100 the proposed approaches outperform the best-performing baseline by 6.40%, 6.09%, and 4.86% under three collaborative learning settings, respectively. Besides, compared with the two upper bound methods, the proposed methods match the performance of the upper bound centralized MoCo under the IID setting and outperform supervised FedAvg on CIFAR-10 under non-IID settings.

5.6.7 Feature Space Clustering

Table 7: Linear evaluation with few labels to evaluate feature space clustering under the IID setting.

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
FedRot	75.12	68.27	34.65	17.24	82.83	75.20
FedDC	76.66	70.47	38.43	22.99	86.76	82.54
FedSwAV	<u>81.43</u>	<u>77.85</u>	<u>46.22</u>	<u>29.52</u>	<u>87.95</u>	83.66
FedSimCLR	81.14	76.74	40.92	27.51	86.55	<u>83.75</u>
FedCA	74.00	64.81	35.09	18.15	83.50	79.30
Proposed	**87.75**	**85.30**	**54.80**	**42.90**	**89.11**	**85.25**
MoCo (Centralized)	87.54	83.48	55.81	39.29	90.27	86.61

5.6.7.1 Linear Evaluation with Different Percentages of Labeled Data

We evaluate the performance of *feature space clustering* of different approaches. In addition to using 100% labeled data for linear evaluation in Table 6, we use different percentages of

Table 8: Linear evaluation with few labels to evaluate feature space clustering under non-IID setting 1 (top) and non-IID setting 2 (bottom).

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
FedRot	71.64	59.97	33.48	17.35	80.75	73.24
FedDC	67.26	54.72	37.95	19.68	83.49	<u>76.24</u>
FedSwAV	<u>71.93</u>	61.32	<u>41.64</u>	<u>23.80</u>	<u>84.73</u>	76.11
FedSimCLR	66.22	56.21	36.58	22.30	81.90	74.06
FedCA	71.47	<u>62.28</u>	34.32	17.40	83.18	75.70
Proposed	**75.05**	**65.20**	**47.75**	**31.33**	**85.59**	**77.30**

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
FedRot	65.80	54.10	31.70	15.85	80.15	70.61
FedDC	66.49	55.30	36.44	20.40	85.93	79.04
FedSwAV	<u>72.99</u>	<u>62.07</u>	<u>44.13</u>	25.40	<u>87.17</u>	<u>79.22</u>
FedSimCLR	68.72	56.21	40.08	<u>25.95</u>	83.59	75.97
FedCA	71.17	61.89	34.24	16.95	83.29	77.15
Proposed	**75.41**	**64.42**	**49.93**	**32.25**	**87.27**	**81.22**

labeled data (10%, 1%) for training the classifier on top of the *fixed* encoder learned with unlabeled data by different approaches. This evaluation metric is inspired by [28], which proposes that in ideal representations, classes are represented by distinct point masses in the feature space, and few labeled data are sufficient to train the classifier. Therefore, with few labeled data for training the classifier, a higher accuracy represents better feature space clustering.

The proposed approaches achieve better feature space clustering than the baselines. The results in the IID setting and two non-IID settings are shown in Tables 7 and 8, respectively. Each table shows the results on three datasets with two labeled ratios. The proposed approaches significantly outperform SOTA with both 10% and 1% labels. Consistent improvements are observed with different labeled ratios, on different datasets, and under different collaborative learning settings.

5.6.8 Semi-Supervised Learning

We further evaluate the proposed approaches by *semi-supervised learning*, where both the encoder and classifier are *finetuned* with 10% or 1% labeled data after learning the encoder on unlabeled data by different approaches. We evaluate the approaches under the IID collaborative learning setting and two non-IID collaborative learning settings. Table 9 shows the comparison of our results against the baselines under the IID collaborative learning setting (top) and non-IID setting 1 (bottom). Our approach significantly outperforms the self-supervised baselines with 10% and 1% labels. Notably, the proposed methods even outperform the upper bound method centralized MoCo on CIFAR-10 and CIFAR-100 datasets under the IID setting.

In addition to the semi-supervised learning results in the IID setting and non-IID setting 1 (Table 9), Table 10 shows the comparison of our results against the baselines under non-IID setting 2. Our approaches outperform the baselines on all three datasets with 10% and 1% labels.

5.6.9 Collaborative Finetuning

We evaluate the performance of the proposed approaches by collaborative finetuning the learned encoder with few locally labeled data on clients. The results under the IID setting, non-IID setting 1, and non-IID setting 2 are shown in Table 11 (top), Table 11 (bottom), and Table 12, respectively. In these collaborative learning settings and three datasets, the proposed approaches consistently outperform the baselines.

Table 9: Semi-supervised learning under IID setting (top) and non-IID setting 1 (bottom).

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
FedRot	85.38	71.62	43.78	19.84	91.23	47.94
FedDC	78.88	44.18	40.69	11.93	88.97	30.25
FedSwAV	84.51	48.96	50.23	13.82	90.48	62.08
FedSimCLR	86.05	75.36	49.54	27.45	91.28	84.46
FedCA	84.15	41.25	48.57	8.13	91.67	36.93
Proposed	**89.27**	**84.79**	**58.49**	**40.71**	**92.18**	**85.63**
MoCo (Centralized)	88.44	81.75	57.76	37.79	92.46	86.78

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
FedRot	77.82	58.48	43.50	18.80	90.80	60.72
FedDC	71.25	31.85	40.85	11.42	86.80	37.91
FedSwAV	78.25	39.87	46.58	14.11	88.77	35.85
FedSimCLR	78.49	58.13	46.89	23.86	90.41	80.72
FedCA	79.75	58.76	48.10	8.07	89.60	38.98
Proposed	**84.01**	**67.87**	**54.85**	**31.29**	**91.29**	**82.13**
MoCo (Centralized)	88.44	81.75	57.76	37.79	92.46	86.78

5.6.10 Transfer Learning

Transfer learning evaluates the generalization of learned features. The features are first learned on the source task in collaborative learning, then evaluated on the target task. Following [16], we train a linear classifier on top of the frozen encoder on the target task. We train the classifier for 500 epochs with Adam optimizer and learning rate 3e-4.

Table 10: Semi-supervised learning in non-IID setting 2. We finetune the encoder and classifier with 10% or 1% labeled data and report the top-1 accuracy.

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
FedRot	81.60	60.54	41.86	17.50	<u>91.31</u>	68.48
FedDC	71.62	30.73	39.51	9.97	88.39	36.26
FedSwAV	78.94	35.74	49.43	11.44	90.01	54.36
FedSimCLR	80.62	<u>60.87</u>	48.90	<u>26.83</u>	91.13	<u>82.02</u>
FedCA	<u>81.28</u>	31.53	<u>50.78</u>	7.87	90.91	21.53
Proposed	**83.52**	**68.38**	**54.78**	**32.08**	**92.23**	**83.92**
MoCo (Centralized)	88.44	81.75	57.76	37.79	92.46	86.78

Two transfer learning tasks are evaluated. In the first task, features are learned on unlabeled CIFAR-10 and then evaluated on CIFAR-100, and in the second task, features are learned on unlabeled CIFAR-100 and then evaluated on CIFAR-10. As shown in Table 13, our approach outperforms all the baselines on both tasks and all collaborative learning settings.

5.6.11 Ablations

5.6.11.1 Effectiveness of Feature Fusion and Neighborhood Matching

We evaluate three approaches. Contrastive learning (CL) is the approach without feature fusion (FF) or neighborhood matching (NM). CL+FF adds feature fusion, and CL+FF+NM further adds neighborhood matching. We evaluate the approaches by linear evaluation and semi-supervised learning (1% labels) under the non-IID collaborative learning setting (non-IID setting 1). As shown in Figure 42, with linear evaluation, CL achieves 74.96% top-1 accuracy.

Table 11: Collaborative finetuning under IID setting (top) and non-IID setting 1 (bottom).

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
FedRot	85.16	74.25	49.97	16.65	90.49	82.81
FedDC	79.98	71.17	42.81	21.47	90.17	84.22
FedSwAV	85.23	<u>78.92</u>	51.67	<u>26.75</u>	91.33	<u>85.10</u>
FedSimCLR	<u>83.52</u>	75.10	<u>51.73</u>	15.32	<u>91.64</u>	84.31
FedCA	82.32	72.77	50.78	21.10	91.57	84.29
Proposed	**89.33**	**82.52**	**56.88**	**33.15**	**92.15**	**87.11**
FedAvg (Supervised)	74.71	39.35	33.16	8.07	87.95	75.68

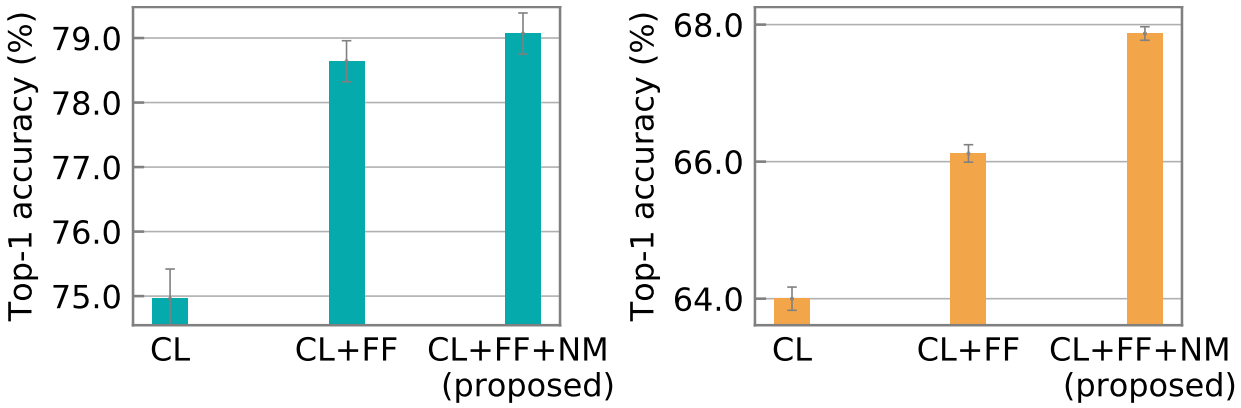
	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
FedRot	57.34	56.80	46.93	17.12	75.02	72.02
FedDC	60.37	49.28	40.29	21.20	77.12	73.16
FedSwAV	57.34	51.93	46.65	<u>22.06</u>	75.45	73.17
FedSimCLR	<u>63.05</u>	51.63	47.69	11.19	<u>76.49</u>	<u>73.25</u>
FedCA	59.52	<u>57.33</u>	<u>49.14</u>	21.50	73.23	71.34
Proposed	**65.80**	**59.30**	**50.75**	**28.25**	**78.81**	**76.88**
FedAvg (Supervised)	48.41	32.33	33.26	8.42	67.42	66.03

Table 12: Collaborative finetuning under non-IID setting 2.

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
FedRot	44.41	28.68	46.04	22.34	60.09	58.75
FedDC	<u>58.05</u>	50.54	39.89	20.25	58.66	58.09
FedSwAV	56.92	48.31	46.97	<u>23.11</u>	51.91	50.32
FedSimCLR	57.34	<u>51.93</u>	47.81	22.37	<u>59.44</u>	<u>58.99</u>
FedCA	42.01	29.21	<u>49.96</u>	21.64	57.76	51.13
Proposed	**62.81**	**53.53**	**51.67**	**25.66**	**63.07**	**65.78**
FedAvg (Supervised)	43.30	29.29	34.37	9.82	54.74	49.76

Table 13: Transfer learning to downstream tasks.

Source	CIFAR-10			CIFAR-100		
Target	CIFAR-100			CIFAR-10		
Distributed setting	IID	non-1	non-2	IID	non-1	non-2
FedRot	40.22	38.56	37.72	68.85	69.34	69.17
FedDC	<u>49.03</u>	<u>45.80</u>	46.25	74.59	73.65	72.97
FedSwAV	49.09	45.73	<u>47.05</u>	74.99	<u>74.94</u>	<u>75.36</u>
FedSimCLR	46.56	39.46	41.28	72.46	71.67	72.83
FedCA	47.43	32.15	47.71	<u>75.15</u>	74.98	75.01
Proposed	**51.43**	**47.90**	**49.49**	**77.28**	**76.37**	**77.50**
Moco (Centralized)	56.74	—	—	80.02	—	—

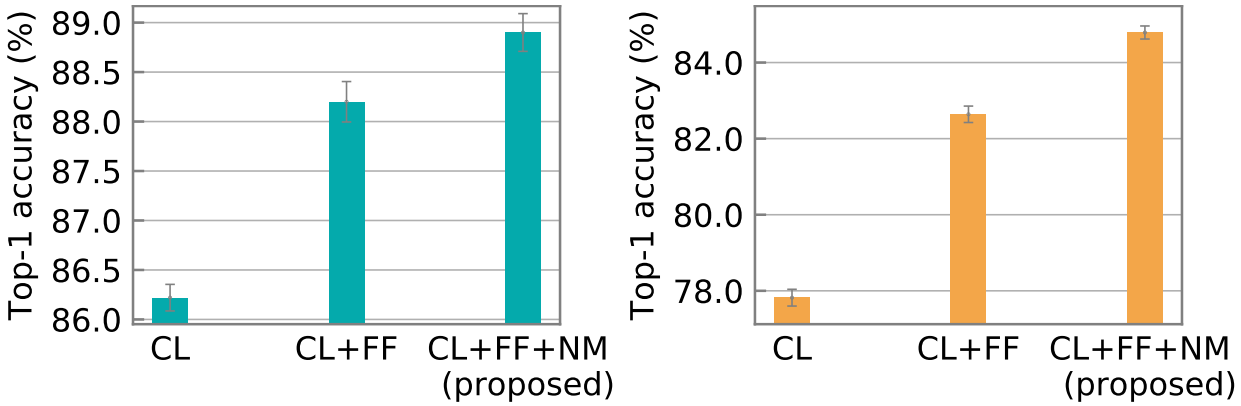


(a) Linear evaluation.

(b) Semi-supervised learning.

Figure 42: Ablations on CIFAR-10 dataset under the non-IID setting.

Adding FF improves the accuracy by 3.68%, and adding NM further improves the accuracy by 0.43%. With semi-supervised learning (1% labels), CL achieves 64.00% top-1 accuracy. Adding FF improves the accuracy by 2.12% and adding NM further improves the accuracy by 1.75%. These results show the effectiveness of feature fusion and neighborhood matching.



(a) Linear evaluation.

(b) Semi-supervised learning.

Figure 43: Ablations on CIFAR-10 dataset under the *IID* setting.

We further perform ablation studies to evaluate the effectiveness of feature fusion and

neighborhood matching under the IID setting. We evaluate three approaches using linear evaluation and semi-supervised learning with 1% labels. As shown in Figures 43 (a) and (b). With linear evaluation, CL achieves 86.22% top-1 accuracy. Adding FF improves the accuracy by 1.98%, and adding NM further improves the accuracy by 0.70%. With semi-supervised learning (1% labels), CL achieves 77.82% top-1 accuracy. Adding FF improves the accuracy by 4.82% and adding NM further improves the accuracy by 2.15%.

Table 14: Impact of image encryption evaluated by linear evaluation under the IID and two non-IID settings.

	CIFAR-10			CIFAR-100			FMNIST		
Method	IID	non-1	non-2	IID	non-1	non-2	IID	non-1	non-2
Without encryption	**89.03**	**81.24**	**78.56**	**62.16**	**59.86**	57.82	91.22	**88.17**	**90.30**
With encryption	88.90	79.07	78.31	61.91	57.54	**58.63**	**91.26**	88.13	90.08

5.6.11.2 Impact of Encrypted Images

We compare the quality of learned representations by using remote features of non-encrypted images for feature fusion, and the results by using remote features of encrypted images for feature fusion. We evaluate the learned representations by four metrics, including linear evaluation, semi-supervised learning, collaborative finetuning, and transfer learning, and the results are shown in Tables 14, 15, 16, and 17, respectively. The results of all four metrics demonstrate that there is very little difference in performance when using features of encrypted versus non-encrypted images, indicating the effectiveness of the proposed approaches in utilizing both types of images.

5.6.12 Learning Curve

We evaluate the learning curve by linear evaluation in the learning process on CIFAR-10. The learning curve represents the learning speed of each approach. We train a linear classifier

Table 15: Impact of image encryption evaluated by semi-supervised learning under IID setting (top), non-IID setting 1 (middle), and non-IID setting 2 (bottom).

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
Without encryption	88.93	83.83	**58.77**	38.90	**92.22**	85.26
With encryption	**89.27**	**84.79**	58.49	**40.71**	92.18	**85.63**

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
Without encryption	**84.72**	**69.66**	**55.26**	**32.97**	91.22	81.93
With encryption	84.01	67.87	54.85	31.29	**91.29**	**82.13**

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
Without encryption	83.48	**68.98**	53.38	29.84	92.17	83.47
With encryption	**83.52**	68.38	**54.78**	**32.08**	**92.23**	**83.92**

Table 16: Impact of image encryption evaluated by collaborative finetuning under IID setting (top), non-IID setting 1 (middle), and non-IID setting 2 (bottom).

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
Without encryption	88.97	80.64	**57.43**	29.11	**92.39**	86.77
With encryption	**89.33**	**82.52**	56.88	**33.15**	92.15	**87.11**

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
Without encryption	**66.54**	**62.36**	**52.52**	25.29	78.18	76.04
With encryption	65.80	59.30	50.75	**28.25**	**78.81**	**76.88**

	CIFAR-10		CIFAR-100		FMNIST	
Labeled ratio	10%	1%	10%	1%	10%	1%
Without encryption	61.99	52.23	50.19	**23.65**	61.57	59.11
With encryption	**62.81**	**53.53**	**51.67**	25.66	**63.07**	**65.78**

Table 17: Impact of image encryption evaluated by transfer learning to downstream tasks.

Source	CIFAR-10			CIFAR-100		
Target	CIFAR-100			CIFAR-10		
Distributed setting	IID	non-1	non-2	IID	non-1	non-2
Without encryption	**53.78**	**51.45**	**50.55**	**78.58**	**78.12**	**77.65**
With encryption	51.43	47.90	49.49	77.28	76.37	77.50

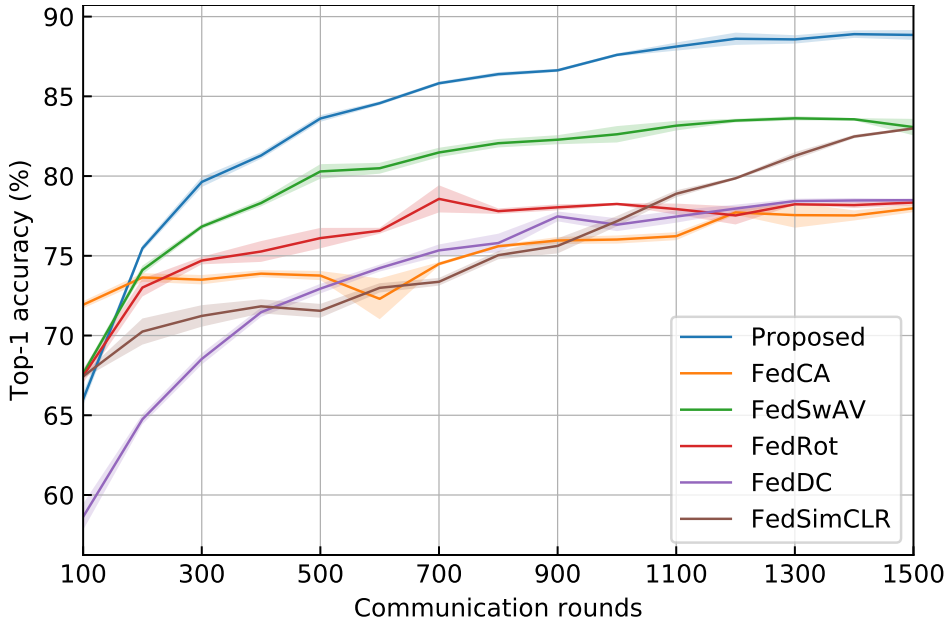
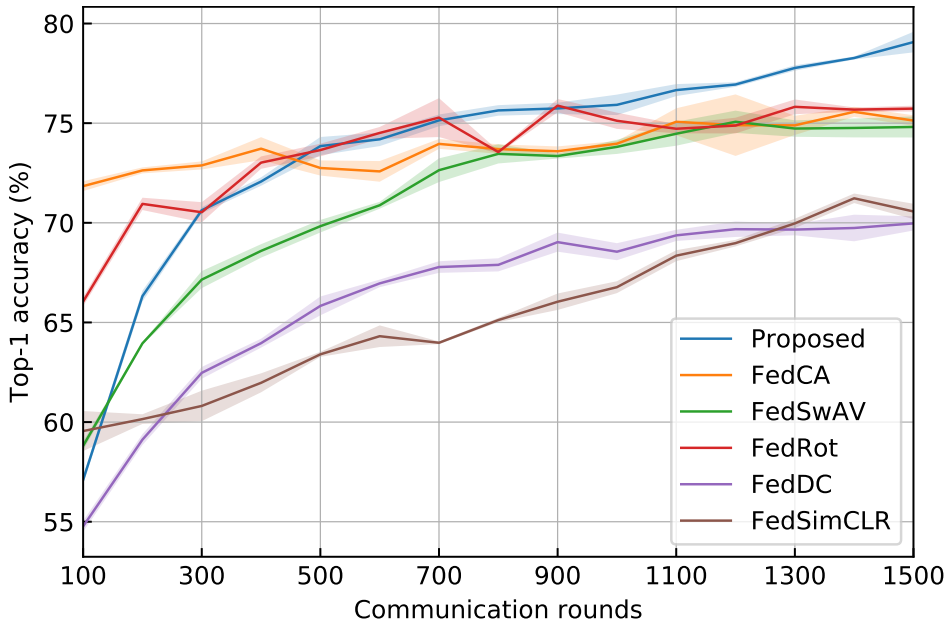


Figure 44: Linear evaluation accuracy against the number of communication rounds on CIFAR-10 in the IID setting.

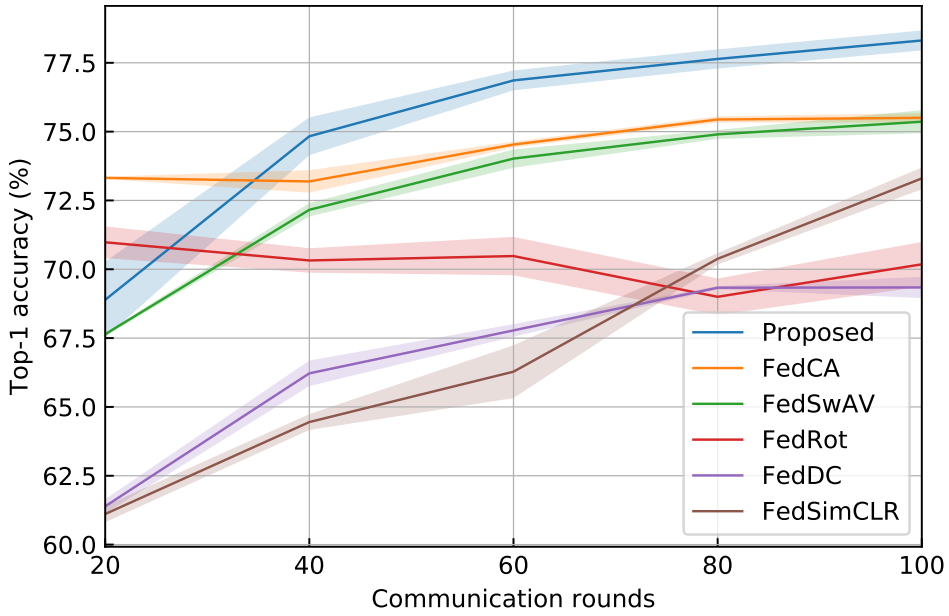
on top of the encoder checkpointed every 100 communication rounds for IID setting and non-IID setting 1, and every 20 rounds for non-IID setting 2. We use classification accuracy as a proxy for representation quality in the learning process.

5.6.12.1 IID Setting

The proposed approaches achieve faster learning and better quality of learned representations than the baselines in both IID and non-IID settings. Figure 44 shows that in the IID setting, the proposed approaches achieve significantly higher linear evaluation accuracy than all baselines after 200 communication rounds. The baseline FedCA achieves slightly higher accuracy in round 100, but it has much lower accuracy in all the following learning processes. This is because FedCA learns from a pre-trained model and can achieve better performance at the initial stage of learning. Different from FedCA, the proposed approaches can learn without any prior knowledge and show superior accuracy over FedCA after the initial stage.



(a) Non-IID setting 1.



(b) Non-IID setting 2.

Figure 45: Linear evaluation accuracy against the number of communication rounds on CIFAR-10 in two non-IID settings.

At the end of the learning process, the proposed approaches achieve an accuracy of 88.90% and outperform the best-performing baseline FedSwAV by 5.28%.

5.6.12.2 Non-IID Setting 1

Figure 45 (a) shows the learning curve in non-IID setting 1. The proposed approaches achieve higher accuracy than all the baselines after 400 rounds. At the end of the learning process, the proposed approaches achieve an accuracy of 79.07% and outperform the best-performing baseline FedRot by 3.19%.

5.6.12.3 Non-IID Setting 2

Figure 45 (b) shows the learning curve in non-IID setting 2. The proposed approaches achieve higher accuracy than all the baselines after 40 rounds. At the end of the learning process, the proposed approaches achieve an accuracy of 78.31% and outperform the best-performing baseline FedCA by 2.81%.

5.7 Summary

We propose a framework for collaborative contrastive representation learning. To improve representation learning on each client, we propose feature fusion to provide remote features as accurate contrastive data to each client. To achieve unified representations among clients, we propose neighborhood matching to align each client's local features to the remote ones. Experiments show superior accuracy of the proposed framework compared with the SOTA.

6.0 Conclusion

This dissertation proposes a framework to enable on-device machine learning inference and training. As edge devices such as IoT devices and mobile devices become ubiquitous, deploying deep learning models, particularly DNNs, on such devices to extract information from the sensed data can enable us to democratize AI. To enable on-device AI, both on-device inference and on-device learning need to be achieved. On-device inference enables edge devices to make predictions based on collected images, such as object classification, to determine the category of objects in the images. On-device training enables devices to learn from their environment and update the model in situ. By applying on-device training to distributed devices, collaborative learning enables a large number of devices to collaboratively learn a shared model while keeping the training data on personal devices to protect privacy.

However, it is challenging to achieve on-device inference and training. First, edge devices usually have limited computation capabilities and limited memory size, but DNNs are usually computationally expensive and require large memory sizes to make predictions. Therefore, DNNs need to be effectively compressed while preserving high accuracy when deploying them to edge devices. Second, DNN training has a high computation cost. There is a significant gap between the high computation and energy demand of on-device training and the limited computing resources and battery life of edge devices. During on-device training, data are collected from the input stream and are unlabeled, which requires learning from new streaming data in-situ with as few labels as possible. Third, during on-device training, each device can only collect a limited amount of data. Model training requires a large amount of data, and training on limited data can result in model overfitting and degrade the model's generalization performance.

To address the challenges of achieving on-device inference and training, this dissertation proposes four techniques. The first challenge, which is the limited on-device computational capability and memory size, is addressed with a proposed model compression framework. This framework effectively compresses a multi-exit neural network through model pruning and quantization, reducing computation cost and model size while preserving accuracy. The

second challenge, the high computation cost of on-device training, is addressed with the proposed efficient training method. This method reduces computation costs by skipping unnecessary training data and pruning gradient computation. Additionally, a data selection approach is proposed to learn data in situ with as few labels as possible, by selecting the most representative data from the input data stream without using labels. The third challenge, the limited amount of data on each device, is addressed with a proposed decentralized unsupervised learning framework. This framework allows multiple distributed devices to collaboratively learn a shared model from decentralized unlabeled data.

Bibliography

[1] Adapting models to the real world: On-device training for edge model adaptation. https://community.arm.com/developer/research/b/articles/posts/adapting-models-to-the-real-world-on-device-training-for-edge-model\-adaptation.

[2] Example on-device model personalization with tensorflow lite. https://blog.tensorflow.org/2019/12/example-on-device-model-personalization.html.

[3] On-device training with core ml. https://machinethink.net/blog/coreml-training-part1/.

[4] Video demo for model compression of multi-exit neural networks: A hard sample. https://youtu.be/IXwi7AWEwac.

[5] Video demo for model compression of multi-exit neural networks: An easy sample. https://youtu.be/gMGs62kOn-s.

[6] Video demo for self-supervised on-device learning framework. https://youtu.be/5ujmt0_MFAI.

[7] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. 2017.

[8] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems*, pages 11816–11825, 2019.

[9] Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, 2017.

[10] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. In *Advances in neural information processing systems*, pages 5145–5153, 2018.

[11] Adrien Bardes, Jean Ponce, and Yann LeCun. VICReg: Variance-invariance-covariance regularization for self-supervised learning. In *International Conference on Learning Representations*, 2022.

[12] Sourav Bhattacharya and Nicholas D Lane. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, pages 176–189, 2016.

[13] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of machine learning and systems*, 1:374–388, 2019.

[14] Zalán Borsos, Mojmir Mutny, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. *Advances in Neural Information Processing Systems*, 33:14879–14890, 2020.

[15] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 132–149, 2018.

[16] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in neural information processing systems*, 33:9912–9924, 2020.

[17] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and M Ranzato. Continual learning with tiny episodic memories. 2019.

[18] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[19] Yiqiang Chen, Xin Qin, Jindong Wang, Chaohui Yu, and Wen Gao. Fedhealth: A federated transfer learning framework for wearable healthcare. *IEEE Intelligent Systems*, 2020.

[20] Alexei Colin and Brandon Lucia. Chain: tasks and channels for reliable intermittent programs. In *ACM SIGPLAN Notices*, volume 51, pages 514–530. ACM, 2016.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[22] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, 27(3):326–327, 1995.

[23] Marc Egger and Detlef Schoder. Consumer-oriented tech mining: Integrating the consumer perspective into organizational technology intelligence-the case of autonomous driving. In *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017.

[24] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough. Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. *Advances in Neural Information Processing Systems*, 32, 2019.

[25] Gao Ge, Wang Chengyan, Zhang Xiaodong, Hu Juan, Yang Xuedong, Wang He, Zhang Jue, and Wang Xiaoying. Quantitative analysis of diffusion-weighted magnetic resonance images: differentiation between prostate cancer and normal tissue based on a computer-aided diagnosis system. *Science China Life Sciences*, 60(1):37–43, 2017.

[26] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[27] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019.

[28] Micah Goldblum, Steven Reich, Liam Fowl, Renkun Ni, Valeriia Cherepanova, and Tom Goldstein. Unraveling meta-learning: Understanding feature representations for few-shot tasks. In *International Conference on Machine Learning*, pages 3607–3616. PMLR, 2020.

[29] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[30] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

[31] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[32] Tyler L Hayes, Nathan D Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9769–9776. IEEE, 2019.

[33] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.

[34] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645, 2016.

[37] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[38] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[39] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Federated visual classification with real-world data distribution. *arXiv preprint arXiv:2003.08082*, pages 76–92, 2020.

[40] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018.

[41] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[42] Yangsibo Huang, Zhao Song, Kai Li, and Sanjeev Arora. Instahide: Instance-hiding schemes for private distributed learning. In *International Conference on Machine Learning*, pages 4507–4518. PMLR, 2020.

[43] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 82–95, 2017.

[44] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[45] Nitthilan Kannappan Jayakodi, Anwesha Chatterjee, Wonje Choi, Janardhan Rao Doppa, and Partha Pratim Pande. Trading-off accuracy and energy of deep inference on embedded systems: A co-design approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11), 2018.

[46] Eunjeong Jeong, Seungeun Oh, Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *arXiv preprint arXiv:1811.11479*, 2018.

[47] Xu Ji, João F Henriques, and Andrea Vedaldi. Invariant information clustering for unsupervised image classification and segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9865–9874, 2019.

[48] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. 2018.

[49] Angela H Jiang, Daniel L-K Wong, Giulio Zhou, David G Andersen, Jeffrey Dean, Gregory R Ganger, Gauri Joshi, Michael Kaminksy, Michael Kozuch, Zachary C Lipton, et al. Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762*, 2019.

[50] Weiwen Jiang, Lei Yang, Edwin H-M Sha, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, Yiyu Shi, and Jingtong Hu. Hardware/software co-exploration of neural architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[51] Weiwen Jiang, Xinyi Zhang, Edwin H-M Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, June. 2019.

[52] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[53] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. Hard negative mixing for contrastive learning. *Advances in Neural Information Processing Systems*, 33:21798–21809, 2020.

[54] Joshua Knights, Ben Harwood, Daniel Ward, Anthony Vanderkop, Olivia Mackenzie-Ross, and Peyman Moghadam. Temporally coherent embeddings for self-supervised video representation learning. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 8914–8921. IEEE, 2021.

[55] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Learning active learning from data. In *Advances in Neural Information Processing Systems*, 2017.

[56] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[57] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10. *CIFAR-10 Dataset*, 2009.

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[59] Yann LeCun. Lenet-5, convolutional neural networks. `http://yann.lecun.com/exdb/lenet/`.

[60] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[61] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[62] Seulki Lee and Shahriar Nirjon. Neuro. zero: a zero-energy neural network accelerator for embedded sensing and inference systems. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, 2019.

[63] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017.

[64] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2(3):429–450, 2020.

[65] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[66] Boyi Liu, Lujia Wang, and Ming Liu. Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems. *IEEE Robotics and Automation Letters*, 4(4):4555–4562, 2019.

[67] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[68] Qing Lu, Weiwen Jiang, Xiaowei Xu, Yiyu Shi, and Jingtong Hu. On neural architecture search for resource-constrained hardware platforms. *arXiv preprint arXiv:1911.00105*, 2019.

[69] Jiahuan Luo, Xueyang Wu, Yun Luo, Anbu Huang, Yunfeng Huang, Yang Liu, and Qiang Yang. Real-world image datasets for federated learning. *arXiv preprint arXiv:1910.11089*, 2019.

[70] Sangkug Lym, Esha Choukse, Siavash Zangeneh, Wei Wen, Sujay Sanghavi, and Mattan Erez. Prunetrain: fast neural network training by dynamic sparse model reconfiguration. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2019.

[71] A. Maxey, C.; Andreas. Measurement and instrumentation data center (midc). *Oak Ridge National Laboratory (ORNL); Rotating Shadowband Radiometer (RSR); Oak Ridge, Tennessee (Data); NREL Report No. DA-5500-56512. http://dx.doi.org/10.5439/1052553*, 2007.

[72] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[73] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[74] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data. 2017.

[75] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[76] Hung T Nguyen, Vikash Sehwag, Seyyedali Hosseinalipour, Christopher G Brinton, Mung Chiang, and H Vincent Poor. Fast-convergent federated learning. *IEEE Journal on Selected Areas in Communications*, 39(1):201–218, 2020.

[77] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016.

[78] Nvidia. High performance ai at the edge: Nvidia jetson tx2. `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/`, 2020.

[79] Emin Orhan, Vaibhav Gupta, and Brenden M Lake. Self-supervised learning through the eyes of a child. *Advances in Neural Information Processing Systems*, 33:9960–9971, 2020.

[80] Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016.

[81] Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. The curious robot: Learning visual representations via physical interactions. In *European Conference on Computer Vision*, pages 3–18. Springer, 2016.

[82] Python. Thop: Pytorch-opcounter. a tool to count the flops of pytorch model., 2020.

[83] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[84] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pages 2021–2031, 2020.

[85] David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International journal of computer vision*, 77(1-3):125–141, 2008.

[86] Ognjen Rudovic, Jaeryoung Lee, Miles Dai, Björn Schuller, and Rosalind W Picard. Personalized machine learning for robot perception of affect and engagement in autism therapy. *Science Robotics*, 2018.

[87] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[88] Stamatios Samaras, Eleni Diamantidou, Dimitrios Ataloglou, Nikos Sakellariou, Anastasios Vafeiadis, Vasilis Magoulianitis, Antonios Lalas, Anastasios Dimou, Dimitrios Zarpalas, Konstantinos Votis, et al. Deep learning on multi sensor data for counter uav applications—a systematic review. *Sensors*, 19(22):4837, 2019.

[89] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the*

IEEE Conference on Computer Vision and Pattern Recognition, pages 4510–4520, 2018.

[90] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

[91] Jahanzaib Shabbir and Tarique Anwer. A survey of deep learning techniques for mobile robot applications. *arXiv preprint arXiv:1803.07608*, 2018.

[92] Qi She, Fan Feng, Xinyue Hao, Qihan Yang, Chuanlin Lan, Vincenzo Lomonaco, Xuesong Shi, Zhengwei Wang, Yao Guo, Yimin Zhang, et al. Openloris-object: A robotic vision dataset and benchmark for lifelong deep learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4767–4773. IEEE, 2020.

[93] Jianzhong Sheng, Chuanbo Chen, Chenchen Fu, and Chun Jason Xue. Easyconvpooling: Random pooling with easy convolution for accelerating training and testing. *arXiv preprint arXiv:1806.01729*, 2018.

[94] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 761–769, 2016.

[95] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[96] Mingcong Song, Kan Zhong, Jiaqi Zhang, Yang Hu, Duo Liu, Weigong Zhang, Jing Wang, and Tao Li. In-situ ai: Towards autonomous and incremental deep learning for iot systems. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018.

[97] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[98] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[99] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016.

[100] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

[101] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[102] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.

[103] Tianchen Wang, Jinjun Xiong, Xiaowei Xu, Meng Jiang, Haiyun Yuan, Meiping Huang, Jian Zhuang, and Yiyu Shi. Msu-net: Multiscale statistical u-net for real-time 3d cardiac mri video segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 614–622. Springer, 2019.

[104] Wenguan Wang, Henry Shu-hung Chung, Ralph Cheng, CS Leung, Xiaoqing Zhan, Alan Wai-lun Lo, J Kwok, Chun Jason Xue, and Jun Zhang. Training neural-network-based controller on distributed machine learning platform for power electronics systems. In *2017 IEEE Energy Conversion Congress and Exposition (ECCE)*, pages 3083–3089. IEEE, 2017.

[105] Yue Wang, Ziyu Jiang, Xiaohan Chen, Pengfei Xu, Yang Zhao, Yingyan Lin, and Zhangyang Wang. E2-train: Training state-of-the-art cnns with over 80% energy savings. In *Advances in Neural Information Processing Systems*, 2019.

[106] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[107] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of*

the IEEE Conference on Computer Vision and Pattern Recognition, pages 10734–10742, 2019.

[108] Yawen Wu, Zhepeng Wang, Zhenge Jia, Yiyu Shi, and Jingtong Hu. Intermittent inference with nonuniformly compressed multi-exit neural network for energy harvesting powered devices. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[109] Yawen Wu, Zhepeng Wang, Yiyu Shi, and Jingtong Hu. Enabling on-device cnn training by self-supervised instance filtering and error map pruning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3445–3457, 2020.

[110] Yawen Wu, Zhepeng Wang, Dewen Zeng, Meng Li, Yiyu Shi, and Jingtong Hu. Decentralized unsupervised learning of visual representations. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI*, pages 2326–2333, 2022.

[111] Yawen Wu, Zhepeng Wang, Dewen Zeng, Yiyu Shi, and Jingtong Hu. Enabling on-device self-supervised contrastive learning with selective data contrast. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 655–660. IEEE, 2021.

[112] Yawen Wu, Zhepeng Wang, Dewen Zeng, Yiyu Shi, and Jingtong Hu. Synthetic data can also teach: Synthesizing effective data for unsupervised visual representation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[113] Yawen Wu, Dewen Zeng, Zhepeng Wang, Yi Sheng, Lei Yang, Alaina J James, Yiyu Shi, and Jingtong Hu. Federated contrastive learning for dermatological disease diagnosis via on-device learning. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–7. IEEE, 2021.

[114] Yawen Wu, Dewen Zeng, Zhepeng Wang, Yiyu Shi, and Jingtong Hu. Federated contrastive learning for volumetric medical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 367–377. Springer, 2021.

[115] Yawen Wu, Dewen Zeng, Zhepeng Wang, Yiyu Shi, and Jingtong Hu. Distributed contrastive learning for medical image segmentation. *Medical Image Analysis*, 81:102564, 2022.

[116] Yawen Wu, Dewen Zeng, Xiaowei Xu, Yiyu Shi, and Jingtong Hu. Fairprune: Achieving fairness through pruning for dermatological disease diagnosis. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2022: 25th International Conference, Singapore, September 18–22, 2022, Proceedings, Part I*, pages 743–753. Springer, 2022.

[117] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018.

[118] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[119] Lei Yang, Weiwen Jiang, Weichen Liu, HM Edwin, Yiyu Shi, and Jingtong Hu. Co-exploring neural architecture and network-on-chip design for real-time artificial intelligence. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 85–90. IEEE, 2020.

[120] Jiecao Yu, Andrew Lukefahr, Reetuparna Das, and Scott Mahlke. Tf-net: Deploying sub-byte deep neural networks on microcontrollers. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):45, 2019.

[121] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pages 12310–12320. PMLR, 2021.

[122] Fengda Zhang, Kun Kuang, Zhaoyang You, Tao Shen, Jun Xiao, Yin Zhang, Chao Wu, Yueting Zhuang, and Xiaolin Li. Federated unsupervised representation learning. *arXiv preprint arXiv:2010.08982*, 2020.

[123] Li Zhang, Haixin Ai, Wen Chen, Zimo Yin, Huan Hu, Junfeng Zhu, Jian Zhao, Qi Zhao, and Hongsheng Liu. Carcinopred-el: Novel models for predicting the carcinogenicity of chemicals using molecular fingerprints and ensemble learning methods. *Scientific Reports*, 7(1):2118, 2017.

[124] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[125] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. In *International Conference on Learning Representations*, 2017.

[126] Nanyang Zhu, Xu Liu, Ziqian Liu, Kai Hu, Yingkuan Wang, Jinglu Tan, Min Huang, Qibing Zhu, Xunsheng Ji, Yongnian Jiang, et al. Deep learning for smart agriculture: Concepts, tools, applications, and opportunities. *International Journal of Agricultural and Biological Engineering*, 2018.