An Efficient and Robust Cloud-Based Deep Learning With Knowledge Distillation

Zeyi Tao, *Student Member, IEEE*, Qi Xia, *Student Member, IEEE*, Songqing Cheng, *Senior Member, IEEE*, and Qun Li, *Fellow, IEEE*

Abstract—In recent years, deep neural networks have shown extraordinary power in various practical learning tasks, especially in object detection, classification, natural language processing. However, deploying such large models on resource-constrained devices or embedded systems is challenging due to their high computational cost. Efforts such as model partition, pruning, or quantization have been used at the expense of accuracy loss. Knowledge distillation is a technique that transfers model knowledge from a well-trained model (teacher) to a smaller and shallow model (student). Instead of using a learning model on the cloud, we can deploy distilled models on various edge devices, significantly reducing the computational cost, memory usage and prolonging the battery lifetime. In this work, we propose a novel neuron manifold distillation (NMD) method, where the student models imitate the teacher's output distribution and learn the feature geometry of the teacher model. In addition, to further improve the cloud-based learning system reliability, we propose a confident prediction mechanism to calibrate the model predictions. We conduct experiments with different distillation configurations over multiple datasets. Our proposed method demonstrates a consistent improvement in accuracy-speed trade-offs for the distilled model.

Index Terms—Deep learning, transfer learning, knowledge distillation

1 Introduction

Recently, deep neural networks (DNNs) [1] have achieved state-of-the-art accuracy on many tasks ranging from computer vision to natural language processing. People usually run the powerful DNNs on the cloud or the proximal edge server [2]. Users hence, execute machine learning (ML) tasks on mobile devices by sending the data such as images, video frames, or speech records to a remote cloud server. Consequently, transmitting a large amount of data uses a lot of network bandwidth, causing network congestion, data plan waste, and lowering the quality of service (QoS). On the other hand, time-sensitive applications such as autopilot, AR games, and control sensors require a quick response between the user and remote server. The network delay or cloud server outages are prohibited and may cause significant loss in such applications.

Practitioners face the dilemma of processing massive data in a local environment or transmitting it to a remote cloud server. Deep learning model deployment is a well-known issue in both the deep learning and edge-cloud community. To increase the model deployability, many practical approaches have been discussed. These methods can be roughly categorized into three types: deep neural network partition [3], [4], [5], model pruning [6], or quantization [7] and knowledge distillation.

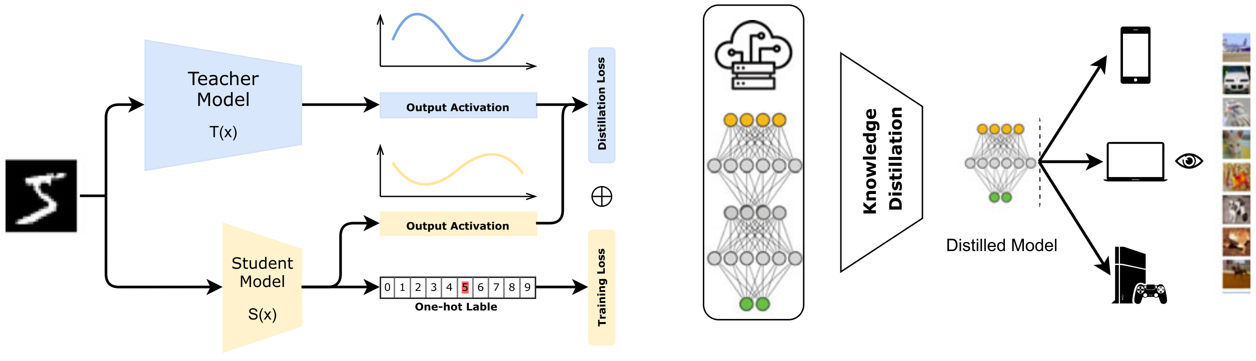
DNNs partition heavily relies on infrastructure support so that we can benefit from executing the model in a parallel manner on different infrastructure components [8]. Although slicing the DNNs layers across the edge increases training parallelism and provides more flexibility in the model placement, it is very challenging to use. Because the problems such as how to discover the optimal model partition strategy, how to deal with the tradeoff between runtime and meta-data transmission make this approach unpopular [9].

Network pruning iteratively prunes the model by removing less important neurons or weights while ensuring the model does not significantly lose accuracy. On the other hand, Parameter quantization tries to decrease the representation precision of weights. Similar to network pruning, a big challenge of the quantized models is preserving high accuracy. Although both network pruning and quantization reduce the computational costs and memory usage of models, they require the pre-knowledge of the model, and people need to retrain and fine-tune the models until they can be fully used [10]. Moreover, the above two approaches are slightly not favored in the edge-cloud community because of the extra training efforts.

In summary, most of the techniques mentioned above are very hard to apply to edge-cloud based ML applications because of two reasons. First, edge-cloud based ML applications are usually running on many different devices and embedding systems. Deploying learning models on heterogeneous edge devices requires lots of effort in searching

- *Zeyi Tao and Qun Li are with Computer Science, College of William and Mary, Williamsburg, VA 23187-8795 USA. E-mail: ztao@email.wm.edu, liqun@cs.wm.edu.*
- *Qi Xia is with Computer Science, William and Mary, Williamsburg, VA 23187-8795 USA. E-mail: qxia01@email.wm.edu.*
- *Songqing Cheng is with Computer Science, George Mason University, Fairfax, VA 22030 USA. E-mail: sqchen@gmu.edu.*

Manuscript received 5 Nov. 2021; revised 31 Jan. 2022; accepted 9 Mar. 2022. Date of publication 22 Mar. 2022; date of current version 7 June 2023.
This work was supported by the U.S. National Science Foundation under grant CNS-1816399 and was also supported by the Commonwealth Cyber Initiative, an investment in the advancement of cyber R&D, innovation and workforce development.
(Corresponding author: Zeyi Tao.)
Recommended for acceptance by D. O. Wu.
Digital Object Identifier no. 10.1109/TCC.2022.3160129



(a) (b)

Fig. 1. (a) Conventional knowledge distillation. A state-of-the-art teacher model (more deeper and wider) displays in blue color and the distilled student model is displayed in cream-coloured. Given input data (handwriting 5), we train the student model via knowledge distillation by two losses. First is a soft loss where we minimize the difference between student activation and teacher activation. This process is also known as student model mimicking teacher model distribution. The second loss is a hard loss where we compare the prediction result to a one-hot label. (b) illustrates a cloud-based machine learning using knowledge distillation. The powerful deep model resides in a cloud server or edge server. While the distilled models can be deployed on the various end devices.

appropriate models, preventing widespread use. Second, deep model reduction techniques such as pruning and quantization spend considerable computation resources in fine-tune training, which is less efficient.

Recently, the emergence of knowledge distillation [11] has become an appealing approach to address the problem of DNNs model deployment. Knowledge distillation transfers knowledge from a deeper and wider model (teacher model) to a shallow model (student model) with negligible performance loss. Recent advances in natural language processes (NLP) such as BERT [12], which has 12 layers and 110 million parameters, can be distilled to 3 or 6 layers shallow models without performance sacrifice [13]. Consequently, using an efficient student network, users can process the learning tasks on resource-constrained devices. Fig. 1a illustrates a typical knowledge distillation process. The key insight of knowledge distillation is that the student model mimics the output distribution of the teacher model. Students acquire information encoded in the teacher's classifiers to achieve the exact prediction as a teacher does. Fig. 1b illustrates a typical knowledge distillation process.

In this paper, we propose a novel cloud-based machine learning framework that users could process the ML tasks on their mobile devices without using the very deep learning models and unnecessarily sharing the local data to the cloud. We use a novel knowledge distillation method named Neural Manifold Distillation (NMD) to reduce the state-of-the-art model complexity without performance loss. Moreover, to improve the reliability, we design an efficient algorithm that enables users to execute the learning task on the cloud if the local predictions are not convincing.

In our design, as shown in Fig. 1b (Fig. 2 for more details), the state-of-the-art models (teacher model) are trained and stored on the cloud or edge server, e.g., through distributed training. Through knowledge distillation, teacher models produce various student models that meet different task requirements and local configurations. In addition, the student models are also trainable by using a local dataset, which enables users to fine-tune their local model.

The proposed knowledge distillation method, NMD, can efficiently produce and improve the performance of distilled student models such that we can execute machine learning tasks on mobile devices with a low computational cost. A key idea in our proposed knowledge distillation method is using feature manifold to transfer knowledge from the teacher model to the student model. The feature manifold is a low-dimensional representation of the original feature space in DNNs. Ideally, we can use this low-dimensional feature representation to recover the original feature space. Our method matches the feature manifolds from the same feature space of the teacher and student models. Compared with the existing knowledge distillation methods, feature manifold gains knowledge from both feature and its geometric information. With additional knowledge, we can improve the performance of student models. Overall, our approach has two advantages. First, our proposed method introduces minimal computation cost without significant performance loss on the distilled model compared with other methods. The overhead that is introduced by generating feature manifolds is negligible. Second, since our method uses additional feature information to help the distillation, we improve the performance of distilled models.

In summary, the contributions of this paper are threefold:

- We propose a new knowledge distillation method that effectively addresses the problem of deep neural network deployment. In particular, our method shows advantages in computation and communication efficiency.
- To further ensure the reliability of the distilled model, we introduce a simple but efficient prediction mechanism for validating the final results.
- Based on the above, we also propose an end-to-end framework for cloud-based machine learning applications running on resource-constrained mobile devices. To the best of our knowledge, this is the first demonstration of knowledge distillation for an edge-cloud environment.

2 Related Work

In this section, we first briefly summarize recent works for accelerating the deep learning in the edge computing. Then we introduce related works about various knowledge distillation techniques.

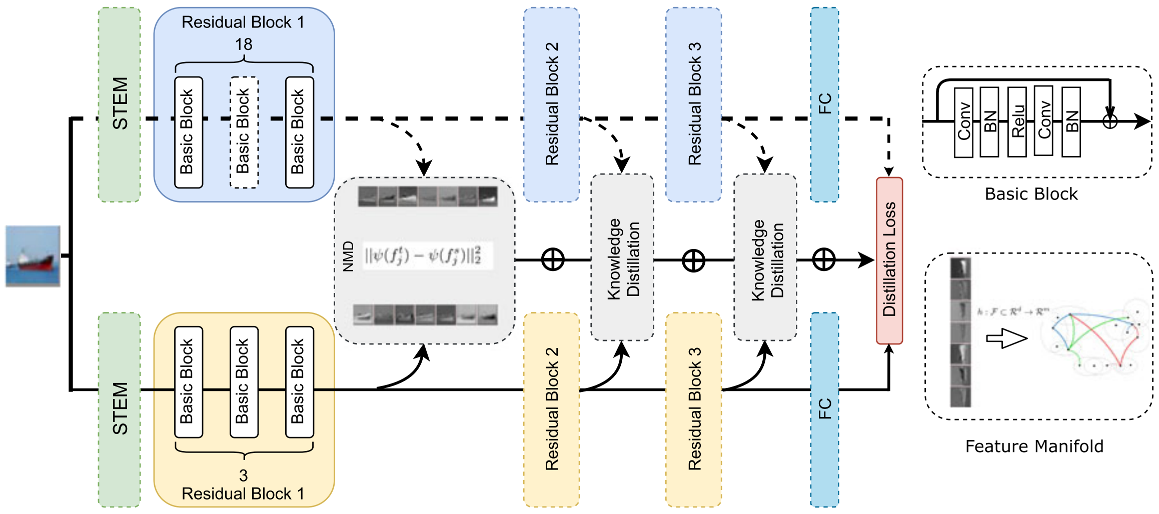


Fig. 2. Overview of our proposed method. The teacher network in light blue is a standard deep residual network [40] (ResNet110). The student network in cream color is a shallow network (ResNet20). Both networks have 3 residual blocks but are different in the number of basic blocks. ResNet110 has 18 basic blocks in each residual block, but ResNet20 only has 3. Knowledge distillation happens after each residual block.

2.1 Efficient Deep Learning in the Edge

In recent years, edge intelligence is expected to push learning computations from the cloud to the edge [14], [15], [16], thus enabling various distributed, low-latency, and reliable, intelligent services. Learning tasks are usually computationally intensive and require large memory footprints. Conventional methods such as model pruning and quantization are not specifically designed for the edge [17]. To this end, researchers find that the DL model can be segmented into multiple partitions and then allocated to 1) distributed edge nodes [18], [19], or 2) use collaborative cloud-edge architecture [20], [21], [22]. The most common segmentation method is partitioning the learning model horizontally, i.e., along the end, edge and cloud. Another model segmentation strategy is vertically partitioning [19]. In contrast to horizontal partition, vertical partition fuses layers and partitions them vertically in a grid fashion, and thus divides CNN layers into independent computation tasks. Chen *et al.* [23] explored the decentralized collaboration in heterogeneous edge training. It can be configured into any specialized submodels for dedicated edge tasks given a predefined CNN model structure. The computation cost of the specialized sub-models can be significantly reduced. The challenge lies in how to select the partition points intelligently.

Other common approaches can be summarized as using shared learning computation. The edge users from the same region might request learning tasks for the same object of interest. In this case, it might introduce redundant computation of learning. Cachier [24] proposes to cache the related models to edge servers to minimize expected end-to-end latency and reuse learning models, according to the offline analysis of applications and online estimates of network conditions. To effectively process streaming data, DeepMon [25] and DeepCache [26] reuse the intermediate results of the video frames from CNN layers to reduce the processing latency of continuous vision applications.

2.2 Knowledge Distillation

Recently, extensive works have been proposed to explore various knowledge distillation techniques. The key to understanding knowledge distillation is knowing how to define the knowledge in the student-teacher training paradigm. The existing efforts can be broadly categorized into three types: soften activation knowledge [11], [27], [28], feature knowledge [29], [30], [31], [32], [33], and first-order Jacobian knowledge [34], [35], [36].

Soften activation knowledge was first proposed by [11]. The insight of work is that the knowledge is defined as soft outputs (soften activation) of the teacher network. The soft knowledge converts the discrete label to a continuous distribution where enlarge the learning space. Student models mimic teacher soft activation distribution as their learning outcomes. The main drawback of using soft knowledge is apparent: soften activation only fits classification tasks. However, for example, in a binary classification problem, soften activation is barely helpful in improving the performance.

Most knowledge distillation techniques have adopted feature knowledge, which improves knowledge distillation by using more helpful information other than soften activations. DNNs usually have a strong ability to extract the features from input data via various operations such as convolution, non-linear activation, residual block, etc. This type of knowledge distillation utilizes multiple intermediate features extracted by DNNs to improve the performance of distilled models. [37] proposed attention transfer (AT) by defining the network knowledge as a spatial attention map of input images. A similar work applied in the computer vision field is named Flow of Solution Procedure [38] (FSP). Later, the success of the attention mechanism in NLP also inspires researchers to apply it to knowledge distillation [29]. While the idea of using contrastive representation learning on knowledge distillation also draws the research attention [30].

First-order Jacobian knowledge has been explored by [34]. The authors describe the neural network as a nonlinear function. Hence, knowledge distillation can be regarded as matching the Jacobians of two neural networks by using the first-order approximation of the functions. Our proposed knowledge distillation method is inspired by Jacobian knowledge in [35].

3 Preliminary

This section will briefly introduce the key concept in knowledge distillation, followed by a discussion of current issues of applying the knowledge distillation to large-scaled edge-cloud environments and possible solutions.

3.1 Knowledge Distillation

The conventional use of knowledge distillation is training a shallow student network from a large teacher ensemble. The teacher model activations are used to guide the training process. Consider a classic supervised learning, suppose we have image dataset $\{x_i, y_i\} \in \mathcal{X} \times \mathcal{Y}, i = 1, 2, \ldots, n$ where $x_i \in \mathcal{X}$ is the i-th input image and $y_i \in \mathcal{Y}$ is its class label. Let $\mathcal{T}(\cdot)$ to be the teacher model, and let $p_i^t = \mathrm{softmax}(z_t^i)$ to be its class probability where z_i^t is the output logit for ith image x_i in the teacher model. In addition, we define $\mathcal{S}(\cdot)$ as a student model with class probability $p_i^s = \mathrm{softmax}(z_i^s)$ where z_i^s is the output logit for ith image x_i in the student model. In knowledge distillation, the student network $\mathcal{S}$ is trained to optimize the following loss function:

$$L_{\mathrm{KD}} = \sum_{i \in [n]} \alpha L_{\mathrm{hard}}(p_i^s, y_i) + (1 - \alpha) L_{\mathrm{soft}}(p_i^s, p_i^t), \tag{1}$$

where L_{hard} is a user-specified loss function used for measuring the prediction p_i^s to ground-truth label y_i, and L_{soft} is loss function for calibrating the soft activation loss between teacher p_i^t and student p_i^s respectively. α is the hyper-parameter to balance the hard and soft losses. Particularly, in the seminar work of Hinton *et al.* [11], they use Kullback-Leibler (KL) divergence for L_{soft} and standard cross-entropy (CE) loss for L_{hard}. By defining soften activation as $p_i^t = \mathrm{softmax}(\frac{z_i^t}{\tau})$ and $p_i^s = \mathrm{softmax}(\frac{z_i^s}{\tau})$, Hinton *et al.* distill the knowledge by optimizing the following:

$$L_{\mathrm{KD}} = \frac{1}{2} \sum_{x_i \in \mathcal{X}} \mathrm{CE}(p_i^s, y_i) + \mathrm{KL}(p_i^s, p_i^t), \tag{2}$$

where τ is a hyper-parameter that aims to produce a smooth continuous distribution over all classes. In this setting, the soften activation knowledge is transferred to the distilled student model. Notice that the τ is only used during the training phase. After that, the τ is set to be 1 for inference. Although the knowledge distillation produces shallow, compact, and computationally efficient models, researchers find the presence of the performance gap between teacher and student models. In the next section, we discuss the main drawbacks of knowledge distillation used in cloud-edge computing.

3.2 Issues With Knowledge Distillation

We produce shallow, compact, and computationally efficient models through knowledge distillation, which can process learning tasks on portable devices. However, recent studies have shown that the distilled model suffers from accuracy loss. Tian *et al.* [30] implement 10 different distillation approaches and find that all of them fail to outperform the knowledge distillation baseline. Moreover, recent studies [39] indicates the presence of a generalization gap on student model. Consequently, applying the distilled model to the edge is challenging due to the need for high-quality models. For example, when a user requests the lightweight distilled model from the cloud server, a distilled model is expected to make decent prediction results. In particular, users may use low-resolution data with noise, which makes distilled model hard to predict. Instead, users send the data to cloud servers seeking a better prediction where massive communication costs are introduced and inefficient.

To effectively improve the performance of the distilled model, let us consider over-parameterized deep learning models where the number of model parameters is far more than the data. DNNs can extract millions of features from the input data to dramatically increase their performance. These features are encoded in the output of the different layers and always presented in a high-dimensional feature space.

Therefore, feature-based distillation methods are more appealing. They, however, have two significant drawbacks. First, feature-based distillation can be regarded as conducting a very strong and strict regularization for student models [41]. When student models heavily rely on intermediate features, student models will likely be overfitting and thus poor generalization. The above reminds us that it is essential to decide what appropriate features to include for training the student model. Second, the teacher features contain noise and usually lie in a very high dimensional space. Directly using these features is not only unsafe but also introduces computation overhead. Moreover, either vanilla knowledge distillation or the aforementioned feature-based knowledge distillation does not utilize the information in the feature geometric structures. Here, the feature geometric structure refers to a feature related to or in contrast with other features in a system of representations.

3.3 Proposed Method

In the proposed distillation method, we relax the use of intermediate teacher features. Instead, we use feature manifold, a low dimensional feature representation, to reduce the feature noise and adequately regulate the training. Suppose the teacher's high-dimensional features can be projected into a low-dimensional nonlinear space. Inspired by Whye *et al.* [42] and Zhang *et al.* [43], we can use local tangent space to approximate a low-dimensional feature manifold. The tangent space we used is constructed from a series of neighborhood feature points to preserve the nonlinear geometric information. The feature manifold plays a very crucial role in our proposed method. It has an important characteristic: the original feature space can be recovered from the feature manifold where all key features and the nonlinear relationship are preserved. Meanwhile, the feature manifold contains less noise. Therefore, in the process of knowledge distillation, we simply use feature manifolds to train the student model. The student model is moderately regularized, and the generalization ability is improved. Moreover, compared to feature-based distilled models, our method learns knowledge more efficiently with the same amount of training time and resources. However, it is not trivial to discover the feature manifold from a set of features sampled from deep models, possibly with noise. We will elaborate on our approach in the next section.

4 Feature Manifold Distillation

In this section, we will introduce our proposed knowledge distillation method.

4.1 Overview of NMD

Neuron manifold distillation (NMD) focuses on transferring numerical and geometric knowledge using low-dimensional feature manifolds. Fig. 2 illustrates the design of our proposed method. The key challenge of the proposed method is how to find such feature representation from noise features. To this end, NMD transfers feature manifold, a low-dimensional feature representation, which can best represent the original feature space on numeric and geometric characteristics. NMD has three optimization goals: (1) minimizing standard training loss; (2) minimizing student model output loss, which is the same as the conventional knowledge distillation; (3) minimizing the feature manifold distillation loss, which occurs after each residual block in ResNet for example. As a result, our method shows high training accuracy and strong generalization ability on the test data. In summary, we combine the method of local feature geometric extraction and conventional knowledge distillation and propose NMD to distill a flexible, portable, and computation-efficient model.

We first briefly demonstrate the overall optimization goal of our distillation method. Considering knowledge distillation training from teacher model to student model with same batch training data, we have teacher feature set $\mathcal{F}^t$, and student feature set $\mathcal{F}^s$. Given a pre-defined feature manifold extraction function $\psi(\cdot)$, we extract feature $f^t \in \mathcal{F}^t$ and $f^s \in \mathcal{S}^t$ as $\psi(f^t)$ and $\psi(f^s)$ respectively. Then we minimize the l_2 distance of the feature manifolds as $||\psi(f_i^t) - \psi(f_i^s)||_2^2$ for each pair. The optimization problem can be formulated as

$$\begin{aligned} L_{\text{KD}} = &\sum_{i\in[n]} \alpha L_{\text{hard}}(p_i^s, y_i) + \beta L_{\text{soft}}(p_i^s, p_i^t) \\ &+ \sum_{j\in|\mathcal{F}^t|} \lambda_j ||\psi(f_j^t) - \psi(f_j^s)||_2^2, \end{aligned} \tag{3}$$

where α, β, γ_i are hyper-parameters that control the weight of each component. Feature manifold extraction function $\psi(\cdot)$ maps the high-dimensional features to a low-dimensional space where reserves key feature characteristics. The low-dimensional feature manifold contains less noise from stochastic min-batch training. The choices of hyper-parameter λ are very important. We choose a relatively large λ value for shallow-level features, and for deep-level features, we prefer a smaller λ value. In the next section, we carefully explain how to extract the feature geometric information from a given feature space as a feature manifold.

4.2 Linear Feature Manifold

We assume there is a d-dimension feature manifold $\mathcal{M} \subset \mathcal{R}^d$ is embedded in a m-dimension space $\mathcal{R}^m$ such that $d \ll m$. We define a construction function $h(\cdot)$

$$h : \mathcal{M} \subset \mathcal{R}^d \to \mathcal{R}^m. \tag{4}$$

We are given a set of N learning features from DNNs, where $f_i \in \mathcal{R}^m$ are sampled possibly with noise from d-dimension feature manifold, i.e.,

$$f_i = h_f(v_i) + \epsilon_i, \quad i = 1, \ldots, N, \tag{5}$$

where v_i is the feature manifold for ith feature f_i and ϵ_i is error. Our goal is to estimate the unknown lower-dimensional feature vector v_i according to f_i, i.e., $f_i \in \mathcal{R}^m$ is projected to $v_i \in \mathcal{R}^d$. In other words, we would like to represent the learning feature f_i by using a low-dimension vector v_i. Next, we start by showing the linear feature manifold approximation to give some intuition first and then explain the method for a realistic non-linear approximation.

We first assume the feature manifold v_i exists and are sampled to construct the feature $f_i \in \mathcal{R}^m$ by using a d-dimensional affine subspace ($\mathcal{A}$), i.e.,

$$f_i = c + \mathcal{A}v_i + \epsilon_i, \quad i = 1, \ldots, N, \tag{6}$$

where $c \in \mathcal{R}^m$, $v_i \in \mathcal{R}^d$ and $\epsilon_i \in R^m$ is error. $\mathcal{A} \in \mathcal{R}^{m\times d}$ is a matrix forms an orthonormal basis of the affine subspace. For all N features, we have

$$F = ce^\top + \mathcal{A}V + E, \tag{7}$$

where $F = [f_1, f_2, \ldots, f_N]$ including all N features, $V = [v_1, v_2, \ldots, v_N]$, $E = [\epsilon_1, \epsilon_2, \ldots, \epsilon_N]$ and e is an N-dimensional column vector of all ones. Equation (7) illustrates the linear feature manifold construction function.

Our goal is going to find c, $\mathcal{A}$ and V to minimize the approximation error E, i.e.,

$$\underset{c,\mathcal{A},V}{\arg\min}\, ||E||_2^2 = \underset{c,\mathcal{A},V}{\arg\min}\, ||F - (ce^\top + \mathcal{A}V)||_2^2, \tag{8}$$

where $||\cdot||_2$ is the Frobenius norm. Optimization problem in Equation (8) can be solved by singular value decomposition (SVD) [42].

Let $c = \bar{f} = Fe/N$ to be the mean of F of all features, the optimal solution of low-rank matrix $\mathcal{A}V$ is computed by the SVD of $F - \bar{f}e^\top$, i.e.,

$$F - \bar{f}e^\top = U\Sigma Q^\top, \tag{9}$$

and

$$\mathcal{A}V = U_d \Sigma_d V_d^\top, \tag{10}$$

where

$$U_d \in \mathcal{R}^{m\times d}, \Sigma_d \in \mathcal{R}^{d\times d}, V_d \in \mathcal{R}^{d\times N}.$$

Particularly, $\Sigma_d = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_d)$ of the d largest singular values of $F - \bar{f}e^\top$. Remember our goal is to find the low-dimension representation $v_i \in V$ by given its corresponding feature f_i. The optimal linear affine transformation $\mathcal{A}$ is given by U_d. We have the close form solution of construction function h such as

$$h(V) = \bar{f}e^\top + U_d V^\top. \tag{11}$$

The low-dimensional feature manifold V can be produced as

$$\begin{aligned} V &= U_d^{-1}(F - \bar{f}e^\top) \\ &= \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_d)Q_d^\top. \end{aligned} \tag{12}$$

Usually, the dimension d of learned linear manifold V is chosen such that $\sigma_{d+1} \ll \sigma_d$. In practice, we use a fixed and small d to produce computational efficient results. The above Equation (12) demonstrates the approach of computing linear feature manifold based on linear assumption in Equation (6).

4.3 Feature Manifold in Deep Feature

Recall our goal is learning the low-dimensional representation of features produced by the deep learning model. Previous Section 4.2 illustrates the linear manifold learning. However, this is not the case for deep learning features. Deep learning features are far more complicated and non-linear. [43] points out that the key difficulty for non-linear manifold learning is unorganized dimension reduced data. In other words, the traditional dimension reduction approaches such as PCA or multidimensional scaling (MDS) fail to reconstruct the underlying structural information in the target space. The Equation (12) may not correctly reflect the high-dimensional projection V on low dimension space. To perform manifold learning of deep learning features, we use an approach inspired by [43] and [44].

We assume that deep learning feature space $\mathcal{F}$ is a subset of $\mathcal{R}^m$. The feature $f_i \in \mathcal{F}$ has unknown generating function $h(v)$, $v \in \mathcal{R}^d$. By training deep learning model, we collect a set of N m-dimensional feature vectors $F = [f_1, f_2, \dots, f_N]$, $f_i \in \mathcal{R}^m$ which is generated from the following noise-free form

$$f_i = h(v_i) \quad \forall i \in [N]. \tag{13}$$

Our objective for manifold learning of non-linear deep feature is to reconstruct v_i from the corresponding feature f_i or $h(v_i)$ without explicitly knowing $h(\cdot)$. Assume that the function h is smooth enough, using first-order Taylor expansion at a fixed v, and define a small neighbourhood $\hat{v}$ such that $\{||\hat{v} - v||^2 < \epsilon\}$, ϵ small. Let $h_v = [h_1(v), \dots, h_m(v)]^\top$, we have

$$h(\hat{v}) = h(v) + \nabla_v h(v)^\top (\hat{v} - v) + \mathcal{O}(||\hat{v} - v||^2), \tag{14}$$

where $\nabla_v h_f(v)^\top \in \mathcal{R}^{m\times d}$ is the Jacobian matrix of h at v, who has the form of

$$\mathcal{J}_h(v) = \begin{bmatrix} \partial h^{(1)}/\partial v^{(1)} & \cdots & \partial h^{(1)}/\partial v^{(1)} \\ \vdots & \vdots & \vdots \\ \partial h^{(m)}/\partial v^{(1)} & \cdots & \partial h^{(m)}/\partial v^{(d)}, \end{bmatrix},$$

where $h^{(i)}$ and $v^{(i)}$ indicate the ith element of vector. Let $\mathcal{J}_h(v) = \nabla_v h_f(v)^\top$, we know that the tangent space $\mathcal{K}_v$ of h at v is spanned by this d column Jacobian matrix $\mathcal{J}_h(v)$, which ensure the dimension of v at most d. The Equation (14) is known as the local affine approximation of the function $h(\cdot)$.

Given two neighboring features $f = h(v)$ and $\hat{f} = h(\hat{v})$, we are unable to construct v and $\hat{v}$ in the affine subspace $h(v) + \mathcal{K}_v$ without knowing function h. To this end, let G_v be a matrix forming an orthonormal basis of $\mathcal{K}_v$, we can represent $\nabla_v h(v)^\top (\hat{v} - v) = G_v \nu$ and $\nu = P_v(\hat{v} - v)$ where $P_v = G_v^\top \nabla_v h(v)^\top$. According to the Equation (14) and ignoring the second-order term, we have

$$G_v^\top (h(\hat{v}) - h(v)).G_v^\top \nabla_v h(v)^\top (\hat{v} - v) = \nu. \tag{15}$$

Knowing G_v, we can compute local affine transformation ν using Equation (15). We restate this transformation procedure in [43] in the following.

Remark 4.1 (Local Affine Transformation [43]). Let varibale $f, \hat{f}, v, \hat{v}, \nu, G_v, P_v$ and function h defined above. Assume that the function h is smooth. We perform manifold learning by approximating local transformation P_v first

$$\int dv \int_{\Omega(v)} ||P_v(\hat{v} - v) - \nu|| d\hat{v},$$

where $\Omega(v)$ is the neighborhood of v. For linear alignment, we have $\hat{v} - v = P_v^{-1}\nu \equiv L_v \nu$. This step is equivalent to optimize

$$\int dv \int_{\Omega(v)} ||\hat{v} - v - L_v \nu|| d\hat{v}.$$

Note that the ν can be approximated via $G_v^\top (h(\hat{v}) - h(v))$.

In Remark 4.1 and Equation (15), the key challenge is computing unknown orthonormal basis G_v of $\mathcal{K}_v$. The G_v can be constructed via Locally Linear Embedding (LLE) [44]. Considering a set of feature $\{f_1, f_2, \dots, f_N\}$ of k-nearest neighbor features of f_i including f_i, we let $\mathcal{S}_{f_i} = [f_{i1}, \dots, f_{ik}]$ be a matrix. To obtain G_v, we optimize Equation (8) in the Section 4.2 using local feature $\mathcal{S}_{f_i}$

$$\arg\min_{f, G_i, \nu} ||\mathcal{S}_{f_i} - (fe^\top + G_i \mathcal{V})||_2^2, \tag{16}$$

where $G_i \in \mathcal{R}^{m\times d}$. The optimal f is given by $\bar{\mathcal{S}}_{f_i}$, the mean of k neighbor features. Then G_i is the d-largest left singular vectors of $\mathcal{S}_{f_i}(I - \frac{1}{k}ee^\top)$ (Equation (10)). We locally produce the d-dimensional feature representation $\mathcal{V}_i = [\nu_{i_1}, \nu_{i_2}, \dots \nu_{i_k}]$ for each $\nu_{i_j} = G_i^\top (f_{i_j} - \bar{\mathcal{S}}_{f_i})$.

To reconstruct the feature manifold v with respect to the local geometry determined by ν, we assume

$$v_{i_j} = \bar{v}_i + L_i \nu_{i_j} + \epsilon_{i_j}. \tag{17}$$

Writing it matrix form

$$V_i = \frac{1}{k} V_i ee^T + L_i \mathcal{V}_i + E_i, \tag{18}$$

where $V_i = [v_{i_1}, v_{i_2}, \dots, v_{i_k}]$ and $E_i = [\epsilon_{i_1}, \dots, \epsilon_{i_k}]$. To optimize the local construction error $||E_i||$:

$$\arg\min_{V_i, L_i} ||E_i|| = \arg\min_{V_i, L_i} ||V_i(I - \frac{1}{k}ee^\top) - L_i \mathcal{V}_i||_2. \tag{19}$$

The optimal L_i can be founded

$$L_i = V_i(I - \frac{1}{k}ee^\top)\mathcal{V}_i^+, \tag{20}$$

where $\mathcal{V}_i^+$ is the Moore-Penrose generalized inverse of $\mathcal{V}_i$. Now we have:

$$\min ||E|| = \sum_i ||V_i(I - \frac{1}{k}ee^\top)(I - \mathcal{V}_i^+ \mathcal{V}_i)||_2^2. \tag{21}$$

Our goal is to determine the optimal V in the low-dimension space. Let S_i be a 0-1 selection matrix such that $V_i = VS_i$ and $W = \text{diag}(W_1, \dots, W_N)$ with $W_i = (I - \frac{1}{k}ee^\top)(I - \mathcal{V}_i^+ \mathcal{V}_i)$. We have

$$\min ||E|| = ||VSW||. \tag{22}$$

Now, let us use Lagrange multiplier μ to impose the constraint that $k^{-1}V^\top V = I_d$, we have

$$\mathcal{L}(V,\mu) = VSW - \mu(k^{-1}V^\top V - 1). \tag{23}$$

Solving V is simply taking the derivative of $\mathcal{L}(V,\mu)$ with respect of V.

In next section, we will explain why this feature manifold is closely related to knowledge distillation in the theory.

4.4 Local Affine Approximators of Neural Networks

In this section, we will answer the question that why it is necessary to use teacher feature in knowledge distillation and the question that how it relates to our feature manifold, we recall the Equation (1) in Section 3,

$$L_{\text{kd}} = \alpha L_{\text{hard}}(p^s, y) + (1-\alpha)L_{\text{soft}}(p^s, p^t).$$

We notice that the soft losses contain information about the relationship between different classes as discovered by teacher. By learning from soft losses, the student network inherits such dark knowledge. To see how does the knowledge distillation relate to our local affine approximation, we define a multi-layer teacher neural network $\mathcal{T} : R^{\text{input}} \to R^{\text{output}}$:

$$\mathcal{T} = t_1(t_2(\cdots(t_l))) \quad \text{and} \quad t_l(x) = \sum_{i=1}^{p_l} a_{l,i}\phi_l(\langle w_l, x\rangle + b_l), \tag{24}$$

where ϕ_l is lth operation layer such as ReLU activation, w_l is the hidden weight, b_l is the bias vector, and $a_{l,i}$ is the output weight vector, and p_l indicates the number of hidden neurons. To simplify the analysis, we assume the lth layer of teacher network extracts the corresponding feature according to $t_l(a_{l-1}) = f$ where a_{l-1} is the activations from layer $l-1$. We know the lth layer feature has a construction function $h_f(v) + \epsilon$ with the respect to low dimensional feature space of V. Then we have following result.

Theorem 1 (Local Affine Equivalence Theorem). *Consider the squared error cost function for matching soft targets of two neural networks at lth layer with p features, given by*

$$L_{\text{soft}}(\mathcal{T}_{t_l}(x), \mathcal{S}_{t_l}(x)) = \sum_{i=1}^{p}(\mathcal{T}^i_{t_l}(x) - \mathcal{S}^i_{t_l}(x))^2,$$

where x is the activations from layer $l-1$. and ξ is noise. Let $f_t = \mathcal{T}_{t_l}(x)$ and $f_s = \mathcal{S}_{t_l}(x)$, the $f_t, f_s \in \mathcal{F}$ are the features in feature space $\mathcal{F}$ which are generated by $\mathcal{T}_{t_l}$ and $\mathcal{S}_{t_l}$, respectively. Then matching soft targets of two neural networks is equivalent to match their feature manifold where

$$E\left[\sum_{i=1}^{p}(\mathcal{T}^i_{t_l}(x+\xi) - \mathcal{S}^i_{t_l}(x+\xi))^2\right] = \sum_{i=1}^{p}(\mathcal{T}^i_{t_l}(x) - \mathcal{S}^i_{t_l}(x))^2 + \sigma^2 E\left[\sum_{i=1}^{p}(\nabla_x h^i_t(v) - \nabla_x h^i_s(v))^2\right] + \mathcal{O}(\rho^4), \tag{25}$$

where $\nabla_x h^i_t(v)$ is the Jacobian matrix of h_t at v.

Proof. To prove this, we recall the Section 4.2, the feature manifolds are given by two construction function where $\mathcal{T}_{t_l} = h_t(\hat{v})$ and $\mathcal{S}_{t_l}(x+\xi) = h_s(\hat{v})$ where ξ is small noise. Using first-order Taylor expansion at a fixed v, and a small neighbourhood with $\{||\hat{v} - v||^2 < \rho\}$ and ρ is small, we have

$$\begin{aligned}
E\left[\sum_{i=1}^{p}(\mathcal{T}^i_{t_l}(x+\xi) - \mathcal{S}^i_{t_l}(x+\xi))^2\right] &= E\left[\sum_{i=1}^{p}(h^i_t(\hat{v}) - h^i_s(\hat{v}))^2\right] \\
&= E_\rho\left[\sum_{i=1}^{p}(h^i_t(v) + \nabla_v h^i_t(v)\rho - h^i_s(v) - \nabla_v h^i_s(v)\rho))^2\right] + \mathcal{O}(\rho^4) \\
&= \sum_{i=1}^{p}(h^i_t(v) - h^i_s(v))^2 + E_\rho\left[\sum_{i=1}^{p}\rho^2(\nabla_x h^i_t(v) - \nabla_x h^i_s(v))^2\right] \\
&= \sum_{i=1}^{p}(\mathcal{T}^i_{t_l}(x) - \mathcal{S}^i_{t_l}(x))^2 + \rho^2 E\left[\sum_{i=1}^{p}(\nabla_x h^i_t(v) - \nabla_x h^i_s(v))^2\right].
\end{aligned}$$

We omit $\mathcal{O}(\rho^4)$ at third and last row. □

We notice that the loss function has two components. The first term indicates the conventional loss of knowledge distillation on samples from both teacher and student. And the second regularizer term represents the difference of local affine approximation of the given networks. The final error terms are small for small ρ and can be ignored. From the observation in Theorem 1, we can conclude that it is ill-considered practice for conventional knowledge distillation that it only transfers the raw CNN activation outputs pointwisely to their offspring model. For a complete knowledge distillation method, transfer of the underlying knowledge residing in the infinitely many data points nearby is also obligatory.

5 Distillation Based Cloud Computing

The knowledge distilled model proposed in this work can be widely used for many real-world applications such as image classification, speech recognition, object detection, etc. There is plenty of research focusing on improving the distilled models' accuracy and speed of networks. As a result, the distilled models are good at providing predictions that are correct most of the time. However, they are failing at telling the user when their predictions can be trusted and when they cannot [45]. Consider an application such as an autonomous driving car with a front camera for detecting the pedestrian. Most of the time, the lightweight model can provide confident predictions that tell whether there is a pedestrian or not. But, in some cases, for example, the heavy rain, the viewing condition is poor, and the outline of pedestrians captured by the front camera is not clear. The model may not provide a reliable prediction. In this case, in order to make more confident predictions, we tend to utilize a more robust model that resides in the central cloud.

In this section, we propose an end-to-end solution for the aforementioned issue. We first define the problem in the following.

5.1 Definitions

The problem we address is supervised multi-class classification with neural networks. Let input $x \in X$ and label $y \in Y = \{1, 2, \ldots, C\}$ are random variables that follow a ground truth joint distribution $\pi(x, y) = \pi(y|x)\pi(x)$.

Let $\mathcal{N}$ be a neural network with $\mathcal{N}(x) = (\hat{y}, Z_x)$. The $\hat{y}$ is neural network prediction and $Z_x = (z_{x,1}, \ldots, z_{x,C+1}) \in R^C$ is confidence scores, one for each possible class identity $1, \ldots, K$. Usually, the confidence score are converted into probabilities $\hat{P}_x = \{\hat{p}_{x,1}, \ldots, \hat{p}_{x,C+1}\}$ by using the softmax function such as

$$\hat{p}_{x,i} = \mathrm{softmax}(z_{x,i}) = \frac{\exp(z_{x,i})}{\sum_{c=1}^{C} \exp(z_{x,i})}, \tag{26}$$

each p_i indicates the probability that the input x falls to one of class c. The neural network will generate the final prediction by selecting the largest probability over all probabilities in $\hat{P}_x$

$$\hat{y} = \arg\max_i \hat{P}_x. \tag{27}$$

Ideally, we expect the well-train model can be indicative of the actual likelihood of correctness such as

$$\mathcal{P}(\hat{y} = y | \hat{p} = p) = p \quad \forall p \in [0, 1], \tag{28}$$

where the probability is over the joint distribution. However, in real-world application, achieving perfect estimation is impossible. In most cases, we notice that the non-confident model will produce the class probabilities $p_{x,i}$ and $p_{x,j}$ with $i \neq j$ close to each other. This indicates that the model $\mathcal{N}$ likely to pair the input x to either class $\hat{y} = i$ or class $\hat{y} = j$. We name such model as uncertain model.

For supervised learning, the expected calibration error [46] (ECE) is a good metric to measure the model uncertainty. One notion of model uncertainty is the difference in expectation between prediction and accuracy

$$\mathbf{E}[|\mathcal{P}(\hat{y} = y | \hat{p} = p) - p|], \tag{29}$$

ECE can be approximated via partitioning predictions into M equally-size bins and taking a weighted average of the bins' accuracy and prediction difference, such as

$$\mathrm{ECE} = \sum_{1}^{m} \frac{|B_m|}{n} |\mathrm{Acc}(B_m) - \mathrm{Pred}(B_m)|, \tag{30}$$

where B_m is a set of indices of samples whose prediction probability p_i falls into the interval $I_m = (\frac{m-1}{M}, \frac{m}{M}]$ such as $B_m = \{i : \frac{m-1}{M} \leq p_i \leq \frac{m}{M}\}$. $\mathrm{Acc}(B_m)$ and $\mathrm{Pred}(B_m)$ are defined as

$$\mathrm{Acc}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbf{1}(\hat{y}_i = y_i), \tag{31}$$

and

$$\mathrm{Pred}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i, \tag{32}$$

We use the ECE score to select the distilled models during the knowledge distillation training. The smaller ECE score indicates the model is more confident in its predictions. Unfortunately, we are unable to calculate the ECE score during the inference stage because we have no way to know the ground truth distribution p (or $\mathrm{Acc}(B_m)$) in advance. Once a non-confident model generates two similar predictions $\hat{y}_i$ and $\hat{y}_j$ with their probability scores $\hat{p}_i$ and $\hat{p}_j$ respectively, we are unable to trust this results. A good practice for addressing this issue is utilizing a more powerful and confident model in the cloud. For example, we could send the input data back to the cloud and receive the prediction from the teacher model.

5.2 Confidence Prediction

Non-confident models such as distilled light-weight models in constrained devices may generate ambiguous predictions due to the low-resolution input data or lacking information. To this end, we propose a simple but effective threshold-based algorithm to ensure the prediction is trustworthy. We explain the details in the following two phases. During the knowledge distillation training phase, we monitor the model accuracy, such as top-1 and top-5 scores. We also select the most confident model by calculating the ECE score for each intermediate model and choosing the one with the lowest ECE score. In the inference phase, given an input $x \in X$, the non-confident model $\mathcal{N}$ will generate a series of probabilities stored in a vector $\hat{P}_x = (\hat{p}_{x,1}, \ldots, \hat{p}_{x,C+1}) \in R^C$. Each score $\hat{p}_i$ indicates the probability that the input x belongs to the class i. The final prediction is made according to Equation (27). We define a non-confident model with ambiguous predictions as

$$|\hat{p}_{x,i} - \hat{p}_{x,j}| < \sigma \quad \hat{p}_{x,i}, \hat{p}_{x,j} \in \hat{P}_x \quad i \neq j, \tag{33}$$

where $\sigma \geq 0$ is a small real number. In our proposed method, we use the largest and second largest probabilities to evaluate the model uncertainty such as

$$\begin{gathered} |\hat{p}_{x,i} - \hat{p}_{x,j}| < \sigma^* \cdot s \quad \text{where} \\ \hat{p}_{x,i} = \arg\max_{p_x} \hat{P}_x \quad \text{and} \quad \hat{p}_{x,j} = \arg\max_{p_x} \{\hat{P}_x - \hat{p}_{x,i}\}, \end{gathered} \tag{34}$$

where s is a constant scalar. We use the equation above to evaluate the model uncertainty due to the following reasons. First, consider a classification task with 10 different classes, such as the cifar10 dataset. It has a small output prediction space (a vector with 10 entries). The largest and second-largest probabilities are good enough to represent the model uncertainty. Second, we observe that the difference between the largest and smallest probability is huge, which is not a good indicator of model uncertainty.

We propose our confidence prediction knowledge distillation based learning algorithm in Algorithm 1.

5.3 Confidence Prediction Machine Learning

Fig. 3 illustrates our confidence knowledge distillation-based machine learning framework. A state-of-the-art teacher model is stored on the cloud. We perform knowledge distillation on the cloud server because it is powerful in computation and it contains a large amount of training data. We ensure the distilled student model quality by maximizing its accuracy and minimizing test ECE scores. Then the cloud sends the distilled model to the server in advance, and this model is ready for use.

Once a remote device starts a machine learning task, it will first request the distilled model from the proximal edge server [47]. As we mentioned before, the distilled model

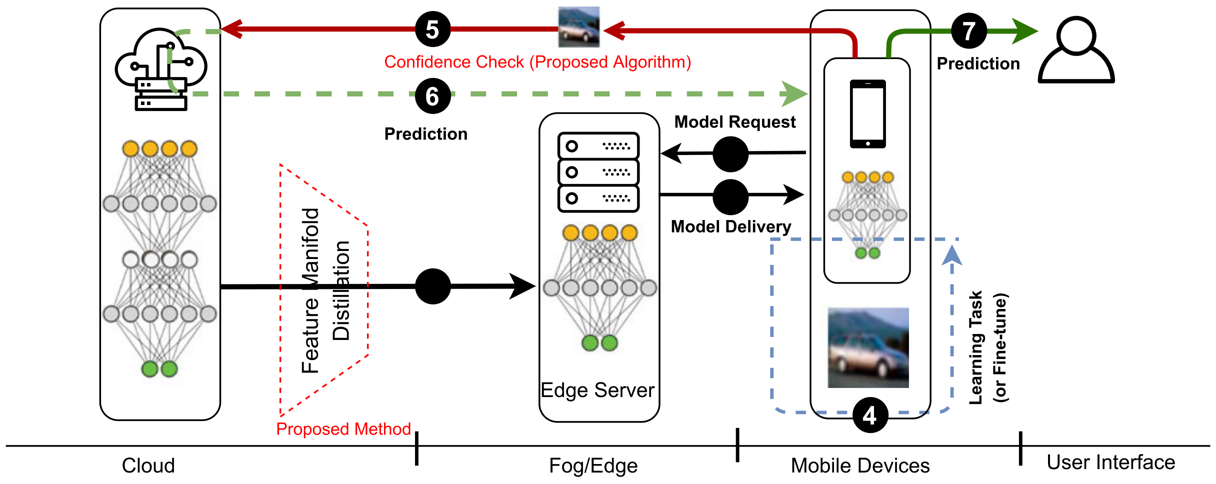


Fig. 3. Knowledge distillation-based machine learning on cloud. The state-of-the-art deep networks deploy on the cloud server. ❶ Use the proposed KD distillation method to produce a shallow and efficient model that meets the different networks configurations and stores them in the edge server. ❷ The user requests a model. ❸ Edge server delivers a model to a user. ❹ User executes the machine learning task on a local device. ❺ If the local model is not confident in the result, send the task to the cloud server.

may not be very confident about its prediction and usually produce ambiguous results. In this case, we first evaluate the prediction results from the distilled model and check if it is an ambiguous result according to the Algorithm 1. If the prediction is ambiguous, we will send the task to the cloud where the powerful teacher network resides to help make predictions (shown in red line and green dash line in Fig. 3).

Algorithm 1. Confidence Prediction on Knowledge Distillation Based Machine Learning

```
Edge Server
1:  Perform knowledge distillation from teacher network $\mathcal{T}$
    to student network $\mathcal{S}$ using algorithm in Section 4.3.
2:  Distilled student model should meet the ECE criteria
    shown in Equation (30), otherwise repeat step 1.
3:  Listening the model request from remote devices.
4:  Listening the prediction request from remote devices.
Remote Device
5:  Request the distilled student model $\mathcal{S}$ from edge server.
6:  Given input data $x$, student model makes prediction vector $\hat{P}_x$
7:  if $|\hat{p}_{x,i} - \hat{p}_{x,j}| < \sigma^* \cdot s$ (Equation (34)) then
8:      Request teacher network prediction on Edge server
        (step 4) and send $x$ to edge server
9:      Return the prediction result $\hat{y}_x$ from teacher model $\mathcal{T}$
10: else
11:     Return the prediction result $\hat{y}_x$ by Equation (27)
12: end if
```

Ideally, the additional prediction help from the teacher model should not be high frequency. We expect the machine learning tasks are processed on a local distilled model other than on the Cloud server. If local models are not confident in prediction, they frequently send the tasks to the cloud server, resulting in a high communication cost. We conduct the communication cost analysis, and experiment details are shown in Section 6.4. In addition, it is worthy of mentioning that the distilled model will be periodically updated by replacing it with a newly generated model from the cloud to keep the model up to date. Furthermore, distilled models can also be trained locally (shown in blue dot line in Fig. 3).

6 Evaluation

In this section, we evaluate the performance of our proposed method on several image classification tasks. We compare our results with conventional knowledge distillation method in [11] (ST). We also compare our results with state-of-the-art distillation methods such as Attention Map Distillation [37] (AT), Attention Features Distillation [29] (AF), Gram Matrix Distillation [38], Sobolev (Jaccobian Matrix) Distillation [48] (SOB), Contrastive Representation Distillation [30] (CRD) etc. We also deploy our distilled model on resource constrained devices such as Raspberry Pi 4 Model B to evaluate the distilled model performance. Raspberry Pi 4 Model B does not have GPU installed, therefore the inference is processed via CPU.

Our experiments cover three significant aspects: (1) the performance of the distilled model, (2) the efficiency of distilled models, and (3) the communication reduction. The first two sets of experiments focus on evaluating the proposed distillation algorithm. The results show that the proposed method achieves promising performance. The last experiment set indicates the efficiency of proposed Algorithm 1 is achieved.

6.1 Experiment Setup

Environment. We conduct our experiments in two different environments. We first compare the distilled student model performance on a single lab server with 4 GeForce GTX 1080ti GPUs. We implement the different knowledge distillation methods by using Pytorch 1.9.0. Next, in order to simulate the cloud-edge environment, we chose a lab server as a cloud server. We perform knowledge distillation on it. Then we use a Raspberry Pi 4 Model B with Quad Core Cortex-A72 (ARM v8) CPU as a remote device. Raspberry Pi 4 Model B does not equip with GPU, therefore the model inference is processed via CPU. Under this experiment setting, we simulate the real-world application that requires the data recall when the model is not confident about its prediction.

Datasets and Deep Neural Networks. We conduct experiments on image classification tasks including CIFAR10,

TABLE 1
The Performance of Different Distillation Methods

TCH/STD Model	KD Method	top-1 Acc.	top-5 Acc.
ResNet110/20	Baseline(ResNet110)	73.15%	95.02%
	ST [11]	63.40%	90.02%
	AT [37]	70.21%	94.36%
	AF [29]	68.48%	96.30%
	Gram matrix [38]	66.10%	93.73%
	SOB [48]	69.01%	94.22%
	CRD [30]	70.61%	95.35%
	NMD (this paper)	**71.92%**	**95.11%**
ResNet110/110	ST	73.10%	95.41&
	AT	73.38%	95.80%
	AF	73.63%	96.30%
	Gram matrix	73.13%	95.43%
	SOB	72.81%	94.12%
	CRD	74.61%	96.35%
	NMD (this paper)	**74.58%**	**96.32%**

TABLE 2
The Training Epoch Time of Different Distillation Methods

TCH/STD Model	KD Method	Time(sec)	Acc.
ResNet20/20	Baseline(ResNet20)	62	92.75%
	ST	72	92.75&
	AT	75	92.45%
	Gram matrix	82	93.8%
	AF	92	93.75%
	SOB	114	93.12%
	CRD	90	93.35%
	NMD (this paper)	88	93.20%

CIFAR100 and ImageNet (ILSVRC2013). The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. CIFAR100, similar to CIFAR10, has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. ImageNet consists of two parts, training data and validation data. The training data contains 1000 categories and 1.2 million images and the validation and test data consists of 150,000 photographs, collected from flickr and other search engines, hand labeled with the presence or absence of 1000 object categories.

The training models we are using include LeNet, AlexNet, ResNet, VGG, GoogleNet and MobileNet. To evaluate the performance of different knowledge distillation methods, we use ResNet110 as teacher model and ResNet20 as student. In our setting, the ResNet110 contains 1.73M trainable parameters and ResNet20 only contains 0.2724M trainable parameters. We roughly reduce the 84% model size in total. We train the model by using a standard SGD optimizer with a start learning rate 0.1 weight decay 0.0001. We train student models within 100 epochs and learning rate decay 10% at epoch 40 and 70.

6.2 Performance Results

We first evaluate the performance of our proposed method and compare it with the-state-of-the-art approaches. We conduct two different types of experiments: knowledge distillation and self-knowledge distillation (as shown in the Table 1). The former one is very common. But in the latter one, we use two exactly the same model as both teacher and student. The self-knowledge distillation aims at testing the distillation algorithm itself only by giving it the same model with the same learning capacity. We use a pre-train ResNet110 model trained on CIFAR10 as a teacher model which provides 73.1% top-1 accuracy and 95.0% top-5 accuracy.

We use ResNet20 as a student model. Both ResNet110 and ResNet20 contain 3 residual blocks but different in the number of layers inside each residual block. For conventional distillation method (ST), we only distill the knowledge from the last layer of the teacher model. For methods such as AT, SOB and CRD, we also distill knowledge from the end of each residual block. For the Gram matrix method, we train the student in two stages. The first stage we initialize the model and train with the teacher model in 50 epochs. Then in the second stage, we conduct normal training on it for another 50 epochs. As shown in Table 1, our proposed method achieves 71.92% accuracy on top-1 result and it also achieves 74.58% accuracy on top-1 with self-knowledge distillation. Notice that our method achieves even better results by comparing it to the baseline model.

In Table 2, we evaluate the single epoch training time of different knowledge distillation approaches. Compared to the normal training process, the extra computation cost comes from 1) data inference time of the teacher model since the teacher model is usually large and the inference time can not be ignored at this time. 2) extra computation of distillation algorithm. For example, the AF method requires building another shallow attention model to extract the attention information from the input intermediate feature which involves a lot of effort on computation. Similarly, the SOB method requires the calculation of gradients. In Table 2, we self-distilled the model ResNet20 on CIFAR10 with batch size 128. The original approach ST used 72s for 1 epoch training. Compared to the baseline, the 10s more epoch time is mainly from teacher model inference.

The methods such as AF, SOB and CRD need more time in computing and matching the different knowledge between teacher model and student model. Our method is 22% slower than the baseline method but we gain $\sim 4\%$ speed up compared to other methods.

We also perform the inference experiments on Raspberry Pi and collect the performance data as shown in Table 3. We train models on the CIFAR100 dataset, and deploy them to Raspberry Pi. The inference time for one epoch mainly depends on the number of parameters of a model. Compared to the other model, VGG16 is the biggest one we deploy on a constrained device and it requires much more computational resources than other ResNets. VGG16 is distilled from another VGG16 network. We see that all the distilled models have better performance than the baseline model which indicates that the knowledge distillation can improve the performance of ultra shallow models. Our proposed method improves 3% accuracy gain on VGG16 and 2% on ResNet34.

6.3 Confidence Prediction

In the Section 5, we discuss the confidence prediction on knowledge distillation. The distilled models are always

TABLE 3
Distilled Models Status on Raspberry Pi

	Model	VGG16	ResN18	ResN20	ResN34
Stats	# para(M)	131	11.4	17.3	21.5
	model size (MB)	478	31	36	40
	GFLOPs (forward)	13.4	1.88	1.92	3.80
	EpochTime (s)	47	9	10	12
Acc.(%)	Baseline	70.48	62.07	66.33	70.02
	ST	71.79	64.33	69.92	72.28
	AT	73.33	63.66	66.87	70.91
	AF	73.41	62.17	69.17	71.39
	CRD	72.93	64.69	70.16	72.58
	NMD	73.40	64.91	69.83	72.66

smaller and have less parameters compared to the teacher model. Therefore, we can not fully trust the prediction results made by a distilled model especially when the input data contains a large amount of noise.

In order to address this problem, we use two strategies to make the distilled model more trustworthy. The first one is we monitor the additional ECE score of the distilled model. Fig. 4a illustrates the ECE score change over the training. The bottom red line indicates the teacher model average ECE score when given different batches of training data. Teacher model ECE score curve is stable. The top blue line is the student model average ECE score. We see the blue line tries to go down at the end of training. But the gap between blue and red curve indicates the model performance difference since the deeper the model is the better the performance is. In fact, it is very hard to mitigate this gap.

We further evaluate the difference of the largest probability score $\hat{p}_{x,i}$ and second largest probability score $\hat{p}_{x,j}$ on inference. Fig. 4c shows that we only have 15% predictions are very confident which means the absolute value between $\hat{p}_{x,i}$ and $\hat{p}_{x,j}$ is close to 1. We see that there are a considerable number of predictions that are quite ambiguous for distilled models, for example those difference scores are less than 0.25. Fig. 4d gives a concrete example on the class of 'plane'. We see that the distilled model has a hard time predicting the 'plane' even at 100 epochs after.

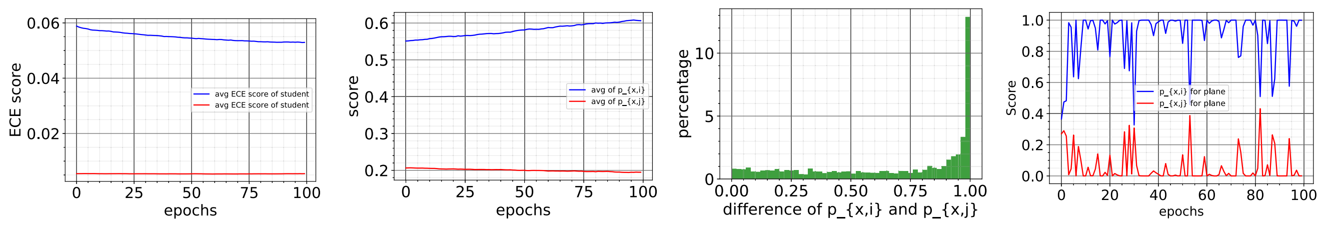
6.4 Communication Reduction

In this section, we evaluate the communication cost for different distillation methods. The major communication reduction is coming from the data transmission to the Cloud server. In our proposed learning framework, users only need to send their local data if the local model is not confident in its predictions. Otherwise, users can trust the local prediction without sending the data to the Cloud server for extra help. As we state in the Algorithm 1, the non-confident predictions are labeled if the difference of largest and second largest probability scores are close enough, say the $\sigma \geq 0$. Given a test batch size b, we will evaluate the predictions on this batch and we define the recall rate as:

$$\text{recall rate } \% = \frac{\sum_{i=1}^{|B|} \mathbf{1}(\{|\hat{p}_{x,i} - \hat{p}_{x,j}| \geq \sigma\})}{|B|} \times 100, \tag{35}$$

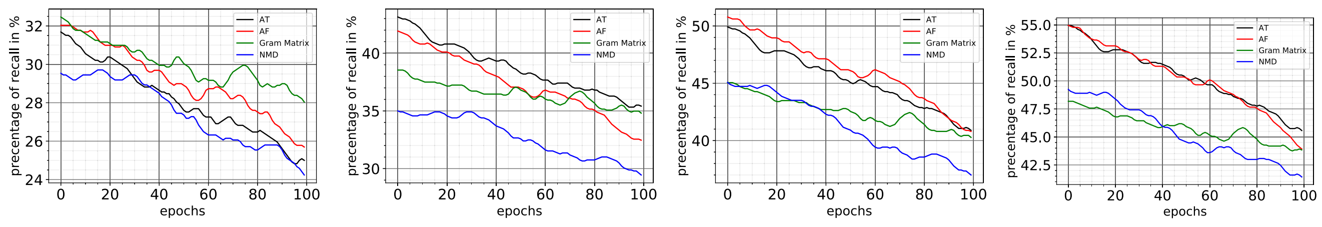
where $\mathbf{1}$ is identity function. Therefore the communication reduction is equivalent to the $1 - \text{recall rate}$ because users do not send that amount of data to Cloud sever.

Fig. 5 illustrates the recall rate of different knowledge distillation methods on different thresholds σ over the training process. We evaluate the result on ResNet20 with the CIFAR10 dataset. The batch size is 128. We have the following observations: (1) the overall recall rates are converging during the training process; (2) our proposed knowledge distillation NMD achieves lowest recall rate for all different



(a) (b) (c) (d)

Fig. 4. We train student network ResNet20 with teacher network ResNet110 on CIFAR10 dataset. We evaluate the model uncertainty by using ECE metric. (a) The average ECE score for student mode (blue) and teacher model (red), we see the method of AT is barely helpful in improving the ECE on student model. (b) We see the discrepancy of largest probability score $\hat{p}_{x,i}$ and the second largest probability score $\hat{p}_{x,j}$ is quite large in average for all classes. However, in (c) we see there are considerable number of probability scores are close to each other during the inference stage. (d) shows the largest probability score $\hat{p}_{x,i}$ and he second largest probability score $\hat{p}_{x,j}$ of class "plane" is hard to be distinguished by student model even after 100 training epochs.



(a) $\sigma = 0.15$ (b) $\sigma = 0.20$ (c) $\sigma = 0.25$ (d) $\sigma = 0.30$

Fig. 5. Recall rate on different knowledge distillation algorithm and different σ(s). Batch size is 128.

thresholds σ at 100 epochs. For example Fig. 5b, when $\sigma = 0.2$, we will return the results of the prediction difference score larger than 0.2. The distilled model trained by our method thinks 29% prediction results should be re-evaluated on the Cloud server. But other methods such as AT, AF and Gram Matrix will send 36%, 35% and 32% respectively to the Cloud server.

7 Conclusion and Future Work

This paper proposes an efficient cloud-edge distributed machine learning framework based on knowledge distillation techniques. We propose a new knowledge distillation method that allows state-of-the-art accuracy for distilled and computation-efficient models. Therefore, we can easily apply the distilled models to resource-constrained devices where users can execute the machine learning tasks in the local environment. To ensure the QoS of distilled models, we present a confidence prediction algorithm to improve the system accuracy.

In future work, we will further improve the efficiency of the distilled model by corporating model pruning and quantization techniques. Furthermore, since an expert selects the current student model, we could further use neural architecture search (NAS) to learn the best model to achieve the efficiency goal. In addition, distilled models should meet different environment requirements, how to fast generate the distilled models should be considered.

Acknowledgments

We thank all reviewers for their helpful comments.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[2] Q. Xia, W. Ye, Z. Tao, J. Wu, and Q. Li, "A survey of federated learning for edge computing: Research problems and solutions," *High-Confidence Comput.*, vol. 1, 2021, Art. no. 100008.

[3] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1802.03835

[4] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "Learning from differentially private neural activations with edge computing," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2018, pp. 90–102.

[5] Y. Mao, W. Hong, H. Wang, Q. Li, and S. Zhong, "Privacy-preserving computation offloading for parallel deep neural networks training," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1777–1788, Jul. 2021.

[6] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. N. Choudhary, and S. Chang, "Fast neural networks with circulant projections," *CoRR*, 2015. [Online]. Available: http://arxiv.org/abs/1502.03436

[7] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[8] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 328–339.

[9] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2224–2287, Jul.–Sep. 2019.

[10] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," Mar. 2015, *arXiv:1503.02531*.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[13] X. Jiao *et al.*, "Tinybert: Distilling bert for natural language understanding," 2019, *arXiv:1909.10351*.

[14] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[15] D. Yu *et al.*, "Decentralized parallel SGD with privacy preservation in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 6, pp. 5211–5220, Jun. 2021.

[16] Z. Xiong, Z. Cai, D. Takabi, and W. Li, "Privacy threat and defense for federated learning with non-iid data in AIoT," *IEEE Trans. Ind. Inform.*, vol. 18, no. 2, pp. 1310–1321, Feb. 2022.

[17] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, Second Quarter 2020.

[18] J. Zhang, S. Chen, B. Liu, Y. Ma, and X. Chen, "A locally distributed mobile computing framework for DNN based android applications," in *Proc. 10th Asia-Pacific Symp. Internetware*, 2018, pp. 1–6.

[19] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018.

[20] G. Li, L. Liu, X. Wang, X. Dong, P. Zhao, and X. Feng, "Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge," in *Proc. Int. Conf. Artif. Neural Netw.*, 2018, pp. 402–411.

[21] Z. Zhao, Z. Jiang, N. Ling, X. Shuai, and G. Xing, "ECRT: An edge computing system for real-time image-based object tracking," in *Proc. 16th ACM Conf. Embedded Netw. Sensor Syst.*, 2018, pp. 394–395.

[22] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the internet of things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan./Feb. 2018.

[23] X. Chen and Z. Qin, "Exploring decentralized collaboration in heterogeneous edge training," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2020, pp. 450–453.

[24] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 276–286.

[25] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile GPU-based deep learning framework for continuous vision applications," in *Proc. 15th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2017, pp. 82–95.

[26] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, "Deepcache: Principled cache for mobile deep vision," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw.*, 2018, pp. 129–144.

[27] R. Müller, S. Kornblith, and G. Hinton, "When does label smoothing help?," 2019, *arXiv:1906.02629*.

[28] B. Heo, J. Kim, S. Yun, H. Park, N. Kwak, and J. Y. Choi, "A comprehensive overhaul of feature distillation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1921–1930.

[29] K. Wang, X. Gao, Y. Zhao, X. Li, D. Dou, and C.-Z. Xu, "Pay attention to features, transfer learn faster CNNs," in *Proc. Int. Conf. Learn. Representations*, 2019.

[30] Y. Tian, D. Krishnan, and P. Isola, "Contrastive representation distillation," 2019, *arXiv:1910.10699*.

[31] K. Yue, J. Deng, and F. Zhou, "Matching guided distillation," in *Proc. 16th Eur. Conf. Comput. Vis.*, 2020, pp. 312–328.

[32] J. Guo, M. Chen, Y. Hu, C. Zhu, X. He, and D. Cai, "Reducing the teacher-student gap via spherical knowledge disitllation," 2020, *arXiv:2010.07485*.

[33] Z. Zhou, C. Zhuge, X. Guan, and W. Liu, "Channel distillation: Channel-wise attention for knowledge distillation," 2020, *arXiv:2006.01683*.

[34] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Swirszcz, and R. Pascanu, "Sobolev training for neural networks," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1706.04859

[35] S. Srinivas and F. Fleuret, "Local affine approximations for improving knowledge transfer," *Idiap*, 2018. [Online]. Available: http://infoscience.epfl.ch/record/256319

[36] Z. Tao, Q. Xia, and Q. Li, "Neuron manifold distillation for edge deep learning," in *Proc. IEEE/ACM 29th Int. Symp. Qual. Service*, 2021, pp. 1–10.

[37] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1612.03928

[38] J. Yim, D. Joo, J. Bae, and J. Kim, “A gift from knowledge distillation: Fast optimization, network minimization and transfer learning,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7130–7138.

[39] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, “Improved knowledge distillation via teacher assistant,” in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 5191–5198.

[40] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[41] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge distillation: A survey,” *CoRR*, 2020. [Online]. Available: https://arxiv.org/abs/2006.05525

[42] Y. W. Teh and S. Roweis, “Automatic alignment of local representations,” in *Proc. 15th Int. Conf. Neural Inf. Process. Syst.*, 2002, pp. 865–872. [Online]. Available: http://dl.acm.org/citation.cfm?id=2968618.2968726

[43] Z. Zhang and H. Zha, “Principal manifolds and nonlinear dimension reduction via local tangent space alignment,” *CoRR*, 2002. [Online]. Available: http://arxiv.org/abs/cs.LG/0212008

[44] S. T. Roweis and L. K. Saul, “Nonlinear dimensionality reduction by locally linear embedding,” *Sci.*, vol. 290, no. 5500, pp. 2323–2326, 2000. [Online]. Available: https://science.sciencemag.org/content/290/5500/2323

[45] L. Neumann, A. Zisserman, and A. Vedaldi, “Relaxed softmax: Efficient confidence auto-calibration for safe pedestrian detection,” *Open Rev.*, 2018.

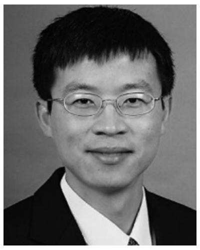
[46] M. P. Naeini, G. Cooper, and M. Hauskrecht, “Obtaining well calibrated probabilities using bayesian binning,” in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 2901–2907.

[47] Z. Tao, Q. Xia, Z. Hao, C. Li, L. Ma, S. Yi, and Q. Li, “A survey of virtual machine management in edge computing,” *Proc. IEEE*, vol. 107, no. 8, pp. 1482–1499, Aug. 2019.

[48] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Świrszcz, and R. Pascanu, “Sobolev training for neural networks,” 2017, *arXiv:1706.04859*.

Zeyi Tao (Student Member, IEEE) is currently working toward the PhD degree with the Department of Computer Science, William & Mary advised by Prof. Qun Li. His research interests include machine learning, deep learning, edge computing, online convex and non-convex optimization, especially the computation and communication efficiency problems in distributed learning systems such as classic distributed machine learning and federated learning.

Qi Xia (Student Member, IEEE) received the BS degree from the University of Science and Technology of China, in 2016, and the PhD degree from the Department of Computer Science, College of William & Mary, in 2021, advised by Prof. Qun Li. His research interests include deep learning, machine learning, edge computing and quantum computing, especially the security and privacy problems in distributed learning systems such as classic distributed machine learning, federated learning and quantum federated learning.



Songqing Chen (Senior Member, IEEE) received the BS and MS degrees in computer science from the Huazhong University of Science and Technology, in 1997 and 1999, respectively, and the PhD degree in computer science from the College of William and Mary, in 2004. He is currently a professor in computer science with George Mason University. His research interests include the Internet content delivery systems, Internet measurement and modeling, system security, and distributed systems. He is a recipient of the U.S. NSF CAREER Award and the AFOSR YIP Award.

Qun Li (Fellow, IEEE) received the PhD degree from Dartmouth College. He is currently a professor with the Department of Computer Science, Wiliam & Mary. His recent research focuses on edge computing and deep learning.