A Resource-Efficient Feature Extraction Framework for Image Processing in IoT Devices

Chuntao Ding, Yidong Li, *Senior Member, IEEE,* Zhichao Lu, Shangguang Wang, *Senior Member, IEEE,* Song Guo, *Fellow, IEEE*

Abstract—Extracting features from image data on Internet of Things (IoT) devices to reduce the amount of data that needs to be uploaded to cloud/edge servers has received increasing attention. However, most of the existing related approaches suffer from two major limitations, (i) low performance and high network traffic, and (ii) a lot of storage resource consumption. To this end, we propose a resource-efficient feature extraction framework for image processing in IoT devices. The proposed framework consists of the edge-assisted extractor generation method and the NestE method. The extractor generated by the edge-assisted extractor generation method can extract the features required by the application, which can not only avoid the IoT device uploading useless feature data but also improve application performance. The proposed NestE generates a nonredundant subextractor by splitting the extractor into multiple subextractors, removing redundant subextractors, and nesting small-capacity subextractors in large-capacity subextractors in a parameter-sharing manner. Compared with deploying multiple independent subextractors on IoT devices, deploying the nonredundant multifunctional extractor can save considerable storage resources and switching overhead. Extensive experimental results show that the proposed framework reduces the storage footprint by approximately 90.7% and switching overhead by approximately 92.4% compared with deploying independent subextractors when using the classical principal component analysis algorithm.

Index Terms—Edge computing, Internet of Things (IoT) devices, feature extraction, resource-efficient.

1 Introduction

The International Data Corporation (IDC) predicts that by 2025, there will be 41.6 billion Internet of Things (IoT) devices, such as smart cameras, Google Glasses, and smartphones, capable of generating 79.4 zettabytes of data [1]. Meanwhile, Gartner predicts that by 2025, 75% of enterprise-generated data will be created and processed outside the cloud server [2]. Therefore, processing data on IoT devices has become mainstream. Since IoT devices are resource (e.g., storage and computing resources) constrained, the collected data are usually processed with the help of the cloud server [3]–[5] or the edge server [6]–[11]. Regardless of cloud-based or edge-based computing paradigms, it is necessary to extract useful data features from the data collected by IoT devices, since the cloud/edge server will further process the received feature data.

Additionally, IoT devices usually run multiple applications simultaneously. The available resources of IoT devices change dynamically due to frequently launching new applications or closing existing applications. When the IoT device starts an application with a higher priority and cannot continue to provide sufficient computing resources for the currently running application; to keep the application running, it needs to switch to a smaller extractor that consumes fewer resources. When the IoT device closes some applications to free available resources, the application can switch to a large extractor that consumes more resources for higher performance. Therefore, to adapt to the dynamically changing available resources of the IoT device, it is necessary to deploy multiple extractors with different sizes and capacities on the IoT device. However, IoT device resources are limited, and deploying multiple extractors will occupy a large amount of IoT device storage resources.

This paper aims to implement a resource-efficient feature extraction framework for IoT devices, with the goal of extracting useful data features on IoT devices and deploying feature extractors in a resource-efficient manner. To this end, this paper first proposes using the dataset information in the edge server to assist in extracting useful features from the data generated by the IoT device. As a dataset of an application, it can provide a benchmark of the features required by the application, which inspires us to explore the dataset information to generate an extractor to assist feature extraction in the IoT device. We first explore the structure information of the dataset to generate an extractor with the ability to extract features. Then, the edge server sends the extractor to the IoT device. Finally, the IoT device uses the extractor to extract the features of the data that meet the needs of the application. With the assistance of the edge server, targeted feature extraction of the data generated by the IoT device can not only improve application performance but also reduce network transmission delay.

Then, this paper proposes the NestE, which splits the generated extractor into multiple subextractors of different capacities. NestE removes redundant subextractors and

- *Chuntao Ding and Yidong Li are with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China 100044. E-mail: {chtding;ydli}@bjtu.edu.cn.*
- *Zhichao Lu is with the School of Software Engineering, Sun Yat-sen University, Zhuhai, 519082, China. E-mail: luzhichaocn@gmail.com.*
- *Shangguang Wang is with the Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China 100876. E-mail: sgwang@bupt.edu.cn.*
- *Song Guo is with the Hong Kong Polytechnic University, Hong Kong, China. E-mail: song.guo@polyu.edu.hk.*
  (Corresponding author: Yidong Li.)

nests these nonredundant subextractors in a parameter-sharing manner to generate a nonredundant multifunctional extractor. The multifunctional extractor can realize the functions of multiple subextractors only by occupying the storage space of the largest nonredundant subextractor. Finally, the edge server sends the multifunctional extractor to the IoT device so that the IoT device can dynamically select the appropriate subextractor based on its currently available resources. Since we use the NestE to remove redundant subextractors and nest the remaining nonredundant subextractors into a single multifunctional extractor, this enables feature extraction on the IoT device in a resource-efficient manner. Note that resource-efficient means that multiple feature extractors consume only a small amount of storage resources and feature extractor switching overhead.

The technical novelty of this paper is to propose a resource-efficient feature extraction framework. The technical depth of this paper is as follows. (i) We address the challenge of extracting features that meet application needs with the help of the dataset in the edge server. (ii) We address the challenge of resource-efficient feature extraction by nesting multiple nonredundant subextractors in a parameter-sharing manner. Experimental results reveal that the proposed framework reduces the storage footprint by approximately 90.7% and reduces switching overhead by approximately 92.4%.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 describes the proposed resource-efficient feature extraction framework in detail. Section 4 presents our evaluation results, and Section 5 concludes the paper.

2 Related Work

To reduce network transmission traffic while considering the resource constraints of IoT devices, many approaches study the useful features of extracting data before uploading the data generated by IoT devices to edge servers. Depending on whether existing methods are independent of extracting features from data collected by IoT devices, we divide them into two categories.

The first category is that the features of data in IoT devices are extracted independently [12]–[16]. For example, Hu *et al.* [12] propose a fog computing-based face identification and resolution scheme. The proposed scheme uses the local binary patterns (LBPs) [17], [18] to extract data features on the fog node, and then the fog node uploads the extracted feature data to the cloud server for feature matching to obtain the recognition result. The proposed scheme saves many bandwidth resources by enabling the fog node to upload only the extracted feature data to the cloud instead of the raw data. Liu *et al.* [16] proposed a food recognition system-based edge computing architecture. The proposed system first uses feature extraction techniques [19]–[21] to preprocess the data collected on the mobile device, and then the mobile device uploads the preprocessed image data to the cloud server and uses deep learning models [22] to process the received data. Finally, the cloud server sends the result to the mobile device. These approaches benefit from offloading the feature extraction process of data to the edge or fog node close to the IoT device so that the edge or fog node only needs to upload the extracted data to the cloud server for processing, which effectively reduces the quantity of transmitted data, thereby reducing the pressure on the core network and the network transmission delay.

However, in these approaches, the data feature extraction in the IoT device is separate from edge/cloud servers, which makes it difficult to extract features required by applications. Different from the above methods, our method generates the feature extractor by utilizing the dataset information on the edge server and uses it to extract features from the data collected by IoT devices. Since the feature extraction of the data in the IoT device is assisted by the edge server, the feature data that meet the application requirements can be extracted.

The second category is to extract useful features of data on the IoT device with the help of the edge server. For example, Wang *et al.* [23] proposed a cloud-guided feature extraction method. The proposed approach first uses discriminative feature extraction techniques [24]–[27] to generate a discriminative feature extractor on the cloud server and then sends the generated extractor to the edge server. After the edge server receives the data uploaded by the IoT device, it first uses the received extractor to extract the discriminative features of the data and then uploads the extracted discriminative feature data to the cloud server for further processing. Ding *et al.* [28] first used discriminative feature extraction techniques to generate a discriminative feature extractor on an edge server and then send the generated extractor to the IoT device. In addition, considering that the available resources of IoT devices are dynamically changing and limited, [28] proposes deploying multiple extractors on IoT devices and proposes the NestDFE method to save resources. Benefiting from device-edge-cloud collaboration, the IoT device can extract the discriminative features of the data, which significantly improves the recognition accuracy and reduces the response time.

However, they ignore the redundancy of the extractors deployed on IoT devices, resulting in a large extractor switching overhead and extractor footprint occupation. Different from the above methods, our method analyzes the extractor in detail and significantly reduces the storage resource overhead and switching overhead by removing redundant extractors.

There is also much work on training or inferring models in a cloud-device collaboration fashion, such as [29]–[36]. Additionally, some excellent work such as [37]–[39] consider task offloading when cloud-device communication is unstable. However, different from the above methods, we focus on how to extract useful features of the data on the IoT device under the constraints of available resources.

3 Detailed design of proposed framework

3.1 Overview

Fig. 1 illustrates the overview of the proposed framework. The proposed framework first uses extractor generation algorithms (*e.g.*, principal component analysis) to explore the structural information of the dataset (*e.g.*, image dataset) on the edge server to generate an extractor $\mathbf{E}$. Then, it uses NestE to divide $\mathbf{E}$ into multiple subextractors, remove

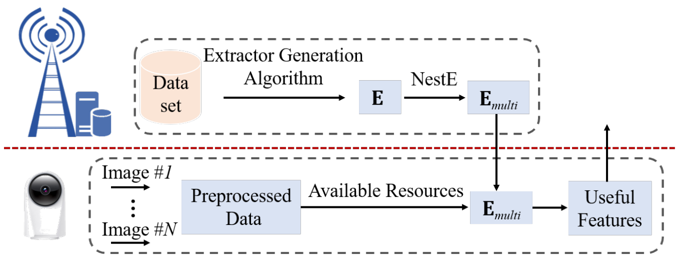


Fig. 1. The proposed framework overview.

redundant subextractors, and nest the remaining nonredundant subextractors into a single extractor to form the multifunctional extractor $\mathbf{E}_{multi}$. Finally, the edge server sends $\mathbf{E}_{multi}$ to the IoT device. After acquiring an image, the IoT device first preprocesses it (*e.g.*, grayscale and dimension alignment), then selects a suitable subextractor (*e.g.*, $\mathbf{S}_{Ei}$) from $\mathbf{E}_{multi}$ by checking its currently available resources to extract useful features from the preprocessed data. Finally, the IoT device sends the extracted feature data to the edge server for further processing.

To clearly show the framework process, we give the offline and the online process, as illustrated in Algorithm 1 and Algorithm 2. The offline process aims to deploy the generated multifunctional extractor $\mathbf{E}_{multi}$ on the IoT device, and the online process aims to use $\mathbf{E}_{multi}$ to process the data collected by the IoT device in real time.

Algorithm 1: Offline Process

```
Input: Datasets $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N \in \mathbb{R}^d$
Output: Nonredundant multifunctional extractor
        $\mathbf{E}_{multi}$
1 Run the feature extractor generation technique to
    obtain $\mathbf{E}$ in the edge server;
2 Run NestE by inputting $\mathbf{E}$ to output $\mathbf{E}_{multi}$;
3 Send $\mathbf{E}_{multi}$ to the IoT device;
4 return $\mathbf{E}_{multi}$;
```

Algorithm 2: Online Process

```
Input: Nonredundant multifunctional extractor
       $\mathbf{E}_{multi}$
Output: Features $\mathbf{v}_i$
1 The IoT device preprocesses the image data $\mathbf{x}_i$;
2 The IoT device checks its currently available
    resources and selects the largest subextractor (e.g.,
    $\mathbf{S}_{\mathbf{E}i}^T$) that it can support from $\mathbf{E}_{multi}$;
3 The IoT device obtains features $\mathbf{v}_i$ through $\mathbf{v} = \mathbf{S}_{\mathbf{E}i}^T \mathbf{x}_i$;
4 return Features $\mathbf{v}_i$;
```

3.2 Feature Extractor Design

The goal of the feature extractor $\mathbf{E}$ is to extract the useful features required by the application from the data generated by the IoT devices. Due to the limited computing and storage resources of IoT devices, they generally extract features from the generated data and then upload the extracted feature data to the edge server for further processing (e.g., feature matching). The extracted features not only affect the subsequent processing (*e.g.*, feature matching, application performance), but also affect the occupancy of network bandwidth. Therefore, it is important to extract useful features from the data generated by IoT devices.

There are many excellent algorithms that can be used to extract features of data on IoT devices, from traditional machine learning algorithms (e.g., PCA [40]–[43], LBP [18]) to current deep neural network models (e.g., ResNet [44], DenseNet [45]). However, how these algorithms can be exploited to extract features useful for applications from the data generated by IoT devices remains to be studied.

Here, we take the PCA algorithm as an example to study this. As an algorithm capable of extracting principal component features of data, there are many advantages to applying the PCA algorithm on IoT devices. For example, unlabeled data are the most common in practical application scenarios, labeled data are time-consuming and labor intensive, and the PCA algorithm can extract the features that best represent the data from unlabeled data, which also plays a role in denoising to a certain extent. Additionally, the PCA algorithm is simple to operate, there is no need to manually set any parameters, and the proof is shown in Theorem 1. The operation of directly using PCA is as follows: The IoT device first uses PCA and the collected data to generate a feature extractor $\mathbf{E}_r$; according to Theorem 1, the value of r can be determined, where r represents the number of positive eigenvalues of the eigendecomposition of the objective function. Then, the IoT device utilizes $\mathbf{E}_r$ to extract the features from the collected data and uploads the extracted feature data to the edge server. However, it is difficult to extract useful features directly using the PCA algorithm. This is because if the PCA method is used directly on the IoT device, the data feature extraction process on the IoT device is made independent. Due to the lack of application guidance, it is difficult for the extracted features to meet the needs of the application. This motivates us to study how to extract the useful features required by applications in IoT devices.

In general, for a type of application, such as image recognition tasks, a given dataset contains all categories of images within the recognizable range. If we want to recognize an image, we generally first use feature extraction methods to extract the features of the image to be recognized and then compare it with the image data in the dataset and select the image label that is most similar to the image data to be recognized in the dataset as the recognition result. This shows that, as an application dataset, it can provide a benchmark of the principal component features required by the application, which inspires us to explore the information of the dataset to generate a feature extractor to assist the feature extraction in IoT devices.

The solution we adopted is to first use the dataset information stored on the edge server and the PCA method to generate a feature extractor $\mathbf{E}$. Formally, given a dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N \in \mathbb{R}^d$, N is the number of images, and d is the dimension of the images. The objective function of PCA is as follows:

$$\begin{aligned} \mathbf{E} = \arg\max_{\mathbf{E}} \{\mathbf{E}^T \mathbf{C} \mathbf{E}\} \\ s.t.\ \mathbf{E}^T \mathbf{E} = \mathbf{I} \end{aligned}, \tag{1}$$

where $\mathbf{C}$ is the covariance matrix and $\mathbf{C} = \frac{1}{N}\mathbf{X}\mathbf{X}^T$. $\mathbf{E}$ is the extractor.

To solve Eq. 1 and facilitate the description of the subsequent NestE method (i.e., Section 3.3), based on [25], [46], [47], we give the following theorem.

Theorem 1. *Assume $\lambda_1 \geq \cdots \geq \lambda_r \geq \cdots \geq 0$ are the eigenvalues of the matrix $\mathbf{C}$ ($\mathbf{C} \in \mathbb{R}^d$), and $\mathbf{E}_1, \cdots, \mathbf{E}_r$ is the pairwise orthogonal unit eigenvectors corresponding to $\lambda_1 \geq \cdots \geq \lambda_r$. The solution to Eq. (1) is composed of eigenvectors $[\mathbf{E}_1, \cdots, \mathbf{E}_r]$.*

Proof. According to the Lagrangian multiplier method, the Lagrangian function of Eq. (1) is:

$$\zeta(\mathbf{E}, \mathbf{\Lambda}) = tr(\mathbf{E}^T\mathbf{C}\mathbf{E}) - tr(\mathbf{\Lambda}(\mathbf{E}^T\mathbf{E} - \mathbf{I})) \ , \tag{2}$$

where $\mathbf{\Lambda} = [\lambda_1, \ldots, \lambda_d]$.

Then, we take the derivative of $\mathbf{E}$ and set it to zero; we have $\mathbf{C}\mathbf{E}_i = \lambda_i \mathbf{E}_i$. Thus, Eq. (1) can be rewritten as:

$$tr(\mathbf{E}^T\mathbf{C}\mathbf{E}) = \sum_{i=1}^{d} \mathbf{E}_i^T\mathbf{C}\mathbf{E}_i = \sum_{i=1}^{d} \mathbf{E}_i^T \lambda_i \mathbf{E}_i = \sum_{i=1}^{d} \lambda_i. \tag{3}$$

Since $\mathbf{C}$ is a positive semidefinite matrix, it has only nonnegative eigenvalues. To maximize $tr(\mathbf{E}^T\mathbf{C}\mathbf{E})$, only the positive eigenvalues should be chosen since zero eigenvalues have no effect on $tr(\mathbf{E}^T\mathbf{C}\mathbf{E})$. Therefore, the solution to Eq. (1) is $\mathbf{E} = [\mathbf{E}_1, \cdots, \mathbf{E}_r]$.

Hence, the statements in this theorem are proved. □

According to Theorem 1, the optimal extractor $\mathbf{E}$ is composed of eigenvectors corresponding to the top r positive eigenvalues, *i.e.*, $\mathbf{E} = [\mathbf{E}_1, \cdots, \mathbf{E}_r]$. The dimension of $\mathbf{E}$ can also be determined to be equal to the number of positive eigenvalues of $\mathbf{C}$. The nature of the extractor composition also shows that PCA is suitable for running on the edge server because it does not require experts to empirically determine the dimensionality of the optimal extractor. Additionally, with the help of the dataset in the edge server, we generate an extractor $\mathbf{E}$ that can extract the useful features required by the application.

3.3 Design of NestE

The obtained extractor $\mathbf{E}$ cannot be directly deployed on the IoT device. This is because, in real-world scenarios, the amount of available resources (*e.g.*, CPU, memory footprint) of the IoT device dynamically changes because it usually runs multiple applications at the same time, new applications are frequently launched, and existing applications are frequently closed. To enable applications on the IoT device to continue to run when the available resources of the IoT device are insufficient, multiple extractors with different capacities need to be deployed. Deploying multiple extractors with different capacities allows IoT devices to flexibly switch from a large-capacity extractor that consumes more resources to a small-capacity extractor that consumes fewer resources to continue running the extractor. Therefore, to adapt to the dynamically changing available resources of the IoT device, it is necessary to split the extractor $\mathbf{E}$ into multiple subextractors with different capacities.

As introduced in Section 3.2, according to Theorem 1, the optimal extractor $\mathbf{E}$ consists of $[\mathbf{E}_1, \cdots, \mathbf{E}_r]$, where r is the number of positive eigenvalues of the objective function. As the dimensionality of the extractor formed by the eigenvectors corresponding to the positive eigenvalues increases, the accuracy also increases. Thus, a straightforward solution is to split $\mathbf{E}$ into r subextractors and deploy them all on the IoT device. That is, we split $\mathbf{E}$ into $\mathbf{S}_{\mathbf{E}1} = [\mathbf{E}_1]$, $\mathbf{S}_{\mathbf{E}2} = [\mathbf{E}_1, \mathbf{E}_2]$, $\mathbf{S}_{\mathbf{E}3} = [\mathbf{E}_1, \mathbf{E}_2, \mathbf{E}_3]$, $\cdots$, and $\mathbf{S}_{\mathbf{E}r} = [\mathbf{E}_1, \mathbf{E}_2, \cdots, \mathbf{E}_r]$ and deploy them all on the IoT device so that the IoT device can choose the appropriate subextractor according to its currently available resources. Note that the larger-capacity subextractor achieves higher accuracy (Section 3.2). Therefore, when the currently available resources allow, the IoT device should select the largest subextractor to obtain the highest accuracy under current conditions. Note that in different application scenarios, users have different requirements for response time and accuracy, and we leave this requirement as one of our future works. Here, we only consider the subextractor that the user prefers to obtain the highest accuracy when available resources allow.

However, through a large number of experiments, we found that the generated extractor $\mathbf{E}$ contains many redundant vectors, as shown in Fig. 5. As the dimensionality of the extractor composed eigenvectors corresponding to the positive eigenvalues increases, as explained in Section 3.2, so does the accuracy. However, subextractors with different dimensions correspond to the same accuracy. As shown in Fig. 5 (a), on the COIL20 dataset, the features extracted by the feature extractor composed of many different numbers of eigenvectors have the same accuracy. Similar results can also be observed in other datasets. We call them redundant subextractors and give the definition of redundant subextractors as follows:

Definition 1. *(Redundant Subextractors) Given two subextractors $\mathbf{S}_{\mathbf{E}i}$ and $\mathbf{S}_{\mathbf{E}j}$, where $\mathbf{S}_{\mathbf{E}i} = [\mathbf{E}_1, \cdots, \mathbf{E}_i]$, $\mathbf{S}_{\mathbf{E}j} = [\mathbf{E}_1, \cdots, \mathbf{E}_j]$, and $i \neq j$, if the features extracted by $\mathbf{S}_{\mathbf{E}i}$ and $\mathbf{S}_{\mathbf{E}j}$ correspond to the same accuracy, $\mathbf{S}_{\mathbf{E}i}$ or $\mathbf{S}_{\mathbf{E}j}$ is redundant.*

In Fig. 5, we observe that it is unwise to directly divide the extractor $\mathbf{E}$ into r subextractors and deploy them on IoT devices because many subextractors are redundant according to Definition 1, which may lead to waste of storage resources, bandwidth resources, and multiple invalid switches. For example, given two redundant subextractors $\mathbf{S}_{\mathbf{E}i}$ and $\mathbf{S}_{\mathbf{E}j}$, $i < j$, the corresponding accuracies of $\mathbf{S}_{\mathbf{E}i}$ and $\mathbf{S}_{\mathbf{E}j}$ are $Acc(\mathbf{S}_{\mathbf{E}i})$ and $Acc(\mathbf{S}_{\mathbf{E}j})$, respectively, $Acc(\mathbf{S}_{\mathbf{E}i}) = Acc(\mathbf{S}_{\mathbf{E}j})$. Since they can obtain the same accuracy, it is only necessary to store $\mathbf{S}_{\mathbf{E}i}$ on the IoT device, and storing both of them will waste IoT device storage resources. Assuming that the quantity of feature data extracted by $\mathbf{S}_{\mathbf{E}i}$ and $\mathbf{S}_{\mathbf{E}j}$ are $Param(\mathbf{S}_{\mathbf{E}i})$ and $Param(\mathbf{S}_{\mathbf{E}j})$, respectively, if $\mathbf{S}_{\mathbf{E}j}$ is used to extract features, it will cause useless data transmission such as $Param(\mathbf{S}_{\mathbf{E}j}) - Param(\mathbf{S}_{\mathbf{E}i})$, resulting in a waste of network resources. In addition, due to the dynamic change in the available resources of IoT devices, for example, the IoT device currently uses $\mathbf{S}_{\mathbf{E}i}$ to extract features from data. At a certain moment, the IoT device closes some applications so that its available resources can support the operation of $\mathbf{S}_{\mathbf{E}j}$ and switches from $\mathbf{S}_{\mathbf{E}i}$ to $\mathbf{S}_{\mathbf{E}j}$ using the principle of selecting the maximum capacity extractor under the available resources to obtain higher accuracy. However, since $Acc(\mathbf{S}_{\mathbf{E}i}) = Acc(\mathbf{S}_{\mathbf{E}j})$, the switching

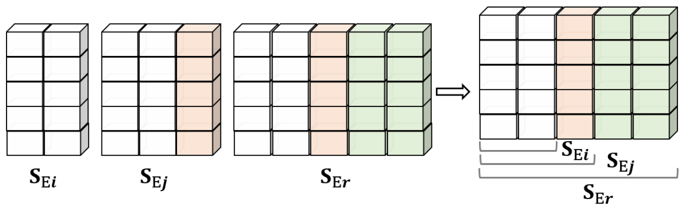

$\mathbf{S}_{\mathrm{E}i}$
$\mathbf{S}_{\mathrm{E}j}$
$\mathbf{S}_{\mathrm{E}r}$
$\mathbf{S}_{\mathrm{E}i}$
$\mathbf{S}_{\mathrm{E}j}$
$\mathbf{S}_{\mathrm{E}r}$

Fig. 2. Illustration of nesting nonredundant subextractors. The small-capacity subextractor is nested in the large-capacity subextractor.

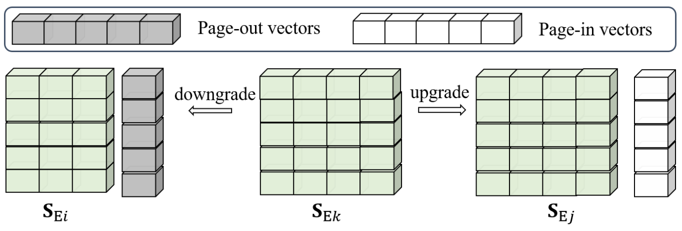

Page-out vectors
Page-in vectors
downgrade
upgrade
$\mathbf{S}_{\mathrm{E}i}$
$\mathbf{S}_{\mathrm{E}k}$
$\mathbf{S}_{\mathrm{E}j}$

Fig. 3. Illustration of the nonredundant multifunctional extractor switching. Note that $\mathbf{S}_{\mathbf{E}i} < \mathbf{S}_{\mathbf{E}k} < \mathbf{S}_{\mathbf{E}j}$.

from $\mathbf{S}_{\mathbf{E}i}$ to $\mathbf{S}_{\mathbf{E}j}$ is invalid, and a certain switching overhead will be generated. It can be seen from the above analysis that it is necessary to remove redundant extractors from the perspective of storage resource consumption, network bandwidth resource consumption, and switching overhead.

Therefore, we first split $\mathbf{E}$ into r subextractors and follow the rule of subextractors composed of $\mathbf{S}_{\mathbf{E}1} = [\mathbf{E}_1]$, $\mathbf{S}_{\mathbf{E}2} = [\mathbf{E}_1, \mathbf{E}_2]$, $\mathbf{S}_{\mathbf{E}3} = [\mathbf{E}_1, \mathbf{E}_2, \mathbf{E}_3]$, $\cdots$, and $\mathbf{S}_{\mathbf{E}r} = [\mathbf{E}_1, \cdots, \mathbf{E}_r]$. Then, we remove redundant subextractors. The subextractors are not independent, that is, the small-capacity subextractor is part of the large-capacity extractor, which provides us with a good split clue to remove redundant subextractors, and we give the definition of removing redundant subextractors.

Definition 2. *(Remove Redundant Subextractors) Given two subextractors $\mathbf{S}_{\mathbf{E}i}$ and $\mathbf{S}_{\mathbf{E}j}$, where $\mathbf{S}_{\mathbf{E}i} = [\mathbf{E}_1, \cdots, \mathbf{E}_i]$, $\mathbf{S}_{\mathbf{E}j} = [\mathbf{E}_1, \cdots, \mathbf{E}_j]$, if $i < j$ and $\mathbf{S}_{\mathbf{E}i}$ and $\mathbf{S}_{\mathbf{E}j}$ correspond to the same accuracy, then remove $\mathbf{S}_{\mathbf{E}j}$.*

According to Definition 2, we can effectively remove redundant subextractors. The next step should be to consider deploying these nonredundant subextractors on resource-constrained IoT devices. The independent deployment of all nonredundant subextractors consumes a large amount of memory space of the IoT device, which makes deployment on IoT devices still a considerable challenge. In addition, this will also consume considerable switching overhead because each time the extractor is switched, the entire subextractor must be paged in and out. For example, when switching a large subextractor (*i.e.*, $\mathbf{S}_{\mathbf{E}j}$) to a small subextractor (*i.e.*, $\mathbf{S}_{\mathbf{E}i}$), the IoT device needs to first page-out $\mathbf{S}_{\mathbf{E}j}$ and then page-in $\mathbf{S}_{\mathbf{E}i}$. Assuming that the number of parameters of $\mathbf{S}_{\mathbf{E}i}$ is $P(\mathbf{S}_{\mathbf{E}i})$ and the number of parameters of $\mathbf{S}_{\mathbf{E}j}$ is $P(\mathbf{S}_{\mathbf{E}j})$, then the number of page-out parameters is $P(\mathbf{S}_{\mathbf{E}j})$, and the number of page-in parameters is $P(\mathbf{S}_{\mathbf{E}i})$. Similarly, when switching from $\mathbf{S}_{\mathbf{E}i}$ to $\mathbf{S}_{\mathbf{E}j}$, the IoT device also needs to first page-out $\mathbf{S}_{\mathbf{E}i}$, and then page-in $\mathbf{S}_{\mathbf{E}j}$, the number of page-out parameters is $P(\mathbf{S}_{\mathbf{E}i})$, and the number of page-in parameters is $P(\mathbf{S}_{\mathbf{E}j})$.

Fig. 2 shows an example where these nonredundant subextractors are not independent and the small-capacity subextractors are part of the large-capacity subextractors. As shown, the vectors represented by the white squares are shared by all subextractors, and vectors represented by the light orange squares are shared by subextractors $\mathbf{S}_{\mathbf{E}j}$ and $\mathbf{S}_{\mathbf{E}r}$. Note that, the number of rows of the subextractor represents the dimension of the input data. This motivates us to explore methods to save storage resources by sharing parameters between subextractors. As shown in Fig. 2, given two subextractors $\mathbf{S}_{\mathbf{E}i}$ and $\mathbf{S}_{\mathbf{E}j}$, where $\mathbf{S}_{\mathbf{E}i} = [\mathbf{E}_1, \cdots, \mathbf{E}_i]$, $\mathbf{S}_{\mathbf{E}j} = [\mathbf{E}_1, \cdots, \mathbf{E}_i, \mathbf{E}_j]$, the subextractor $\mathbf{S}_{\mathbf{E}i}$ is part of the subextractor $\mathbf{S}_{\mathbf{E}j}$. Imagine deploying only one $\mathbf{S}_{\mathbf{E}j}$ on the IoT device and label a part of $\mathbf{S}_{\mathbf{E}j}$ as $\mathbf{S}_{\mathbf{E}i}$. This makes it possible to implement the functionality of multiple subextractors by deploying only one subextractor on the IoT device. Therefore, we propose a NestE algorithm, which nests the small-capacity subextractor into the large-capacity subextractor to form a multifunctional extractor $\mathbf{E}_{multi}$. Note that the multifunctional extractor $\mathbf{E}_{multi}$ can realize the functions of multiple extractors. We present the NestE process in Algorithm 3.

```
Algorithm 3: NestE
  Input: The feature extractor $\mathbf{E}$
  Output: The nonredundant multifunctional
          extractor $\mathbf{E}_{multi}$
1 Split $\mathbf{E}$ into $r$ subextractors, where $\mathbf{S}_{\mathbf{E}1} = [\mathbf{E}_1]$,
    $\mathbf{S}_{\mathbf{E}2} = [\mathbf{E}_1, \mathbf{E}_2]$, $\mathbf{S}_{\mathbf{E}3} = [\mathbf{E}_1, \mathbf{E}_2, \mathbf{E}_3]$, $\cdots$,
    $\mathbf{S}_{\mathbf{E}r} = [\mathbf{E}_1, \mathbf{E}_2, \cdots, \mathbf{E}_r]$;
2 Determine redundant subextractors according to
    Definition 1;
3 Remove redundant subextractors according to
    Definition 2;
4 The small-capacity subextractor is nested into the
    large-capacity subextractor to form the
    nonredundant multifunctional extractor $\mathbf{E}_{multi}$.
    The capacity of $\mathbf{E}_{multi}$ is equal to the capacity of
    $\mathbf{S}_{\mathbf{E}r}$;
5 Return $\mathbf{E}_{multi}$;
```

According to the proposed NestE, when there are multiple nonredundant subextractors, only the subextractor with the largest capacity needs to be deployed on the IoT device to realize the functions of all subextractors. Compared with deploying multiple independent subextractors, deploying only a single subextractor on the IoT device in a parameter-sharing manner significantly reduces storage resource consumption by using the NestE algorithm.

In addition, the multifunctional extractor $\mathbf{E}_{multi}$ can also reduce the overhead of subextractor switching. Fig. 3 illustrates the switching process of the subextractors in $\mathbf{E}_{multi}$. Assuming that the currently running subextractor is $\mathbf{S}_{\mathbf{E}k}$, when the IoT device does not have enough available resources to continue running $\mathbf{S}_{\mathbf{E}k}$ due to the launch of a higher priority application, it needs to switch to a smaller subextractor (*e.g.*, , $\mathbf{S}_{\mathbf{E}i}$) to keep the application running. Note that we use the principle of selecting the largest-capacity extractor under available resources. In this case, the IoT device only needs to page-out the vectors marked

TABLE 1
Description of eight benchmark datasets.

Datasets	# Images	# Dimensions	# Classes	# Train/Test
LEAVES	186	1,024	3	9/1
USPS	1,854	256	10	9/1
YALE	165	1,024	15	7/3
COIL20	1,440	1,024	20	4/6
UMIST	564	1,024	20	3/7
YALEB	2,414	1,024	38	9/1
ORL	400	1,024	40	4/6
COIL	7,200	1,024	100	9/1

as dark squares, the number of page-out parameters is $P(\mathbf{S}_{\mathbf{E}k}) - P(\mathbf{S}_{\mathbf{E}i})$, and the generated page-in parameter is zero. When the IoT device has enough available resources to perform a larger subextractor, it switches the subextractor $\mathbf{S}_{\mathbf{E}k}$ to $\mathbf{S}_{\mathbf{E}j}$ to extract more features to obtain higher accuracy. The IoT device only needs to page-in the vectors marked as white squares, the number of page-in parameters is $P(\mathbf{S}_{\mathbf{E}j}) - P(\mathbf{S}_{\mathbf{E}k})$, and the generated page-out parameter is zero. Compared with deploying independent feature extractors, deploying extractors through parameter sharing significantly reduces the switching overhead.

Therefore, deploying $\mathbf{E}_{multi}$ on the IoT device allows the IoT device to dynamically select subextractors to extract useful features based on its currently available resources. In addition, the nature of $\mathbf{E}_{multi}$ nesting multiple subextractors not only saves the storage space of the IoT device but also reduces switching overhead and improves the switching efficiency of subextractors.

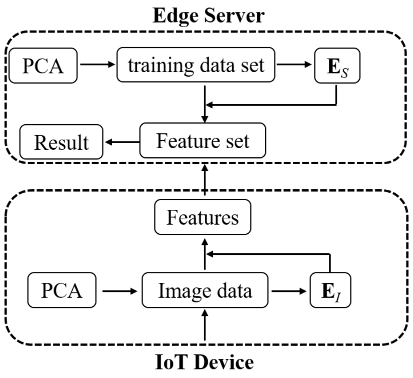
4 Evaluation

In this section, we first introduce our experimental setup, including the datasets, prototype system, baselines, and evaluation metrics studied in this work. Then, we provide an empirical comparison in terms of accuracy, memory space reduction, network traffic, and switching overhead on multiple benchmarks.

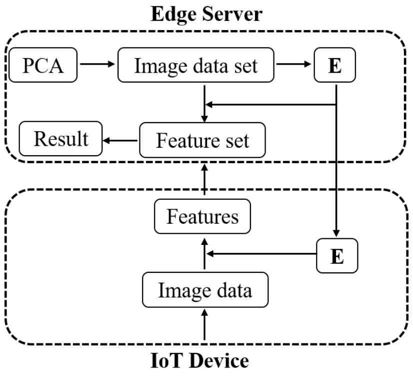
4.1 Experimental Setup

Datasets. To validate the proposed approach, we conduct experiments on eight benchmark datasets: The LEAVES [25] dataset contains 186 images of leaves against different backgrounds. The USPS [48] dataset contains 1,854 grayscale images, which are centered, normalized and show a broad range of font styles. The YALE [49] dataset contains 165 images of 15 subjects. The COIL20 [50] dataset contains 1,440 normalized images of 20 objects. The UMIST [51] dataset consists of 564 images of 20 individuals. Each individual is shown in a series of poses from side to front view. The YAEB [52] dataset contains 2,414 frontal-face images over 38 subjects and approximately 64 images per subject. The images were taken under different lighting conditions and various facial expressions. The ORL [53] dataset contains 400 images with 40 different subjects, and each subject includes 10 different images. The COIL [54] dataset contains images of 100 objects. Table 1 lists the details of datasets used in the experiment.

Prototype System. We build a prototype system to validate the proposed framework. In the proposed system,



(a) Independent framework.



(b) Our proposed framework.

Fig. 4. Independent framework vs. our proposed framework. Note that the greatest difference between the two is that the extractor used in the independent framework to extract features from data on the IoT device is generated on the IoT device, while the extractor in our proposed framework is generated on the edge server.

we use a Huawei honor 8 smartphone equipped with 4 Cortex A72 2.3 GHz and Android 7.0 as a IoT device. We also use Java to invoke OpenCV libraries to preprocess the captured images on Huawei honor 8. A computer equipped with an Intel i7-1165G7@ 2.80 GHz CPU and 16 GB RAM is used as the edge server. The edge server runs Ubuntu 18.04 and implements the PCA and NestE algorithms and feature matching algorithm by Python. The smartphone is connected to the computer via WiFi.

Baselines. To verify the effect of edge server assistance, we compared three frameworks: raw data, independent PCA, and our proposed framework. Using raw data means that the IoT device directly uploads the collected data to the edge server for processing. In addition, we also validate the NestE in our proposed framework using weight adaptive projection matrix Learning (WAPL) [23] and discriminative feature extraction (DFE) [28] algorithms.

Independent PCA refers to using PCA directly to process data collected by the IoT device. As shown in Fig. 4 (a). Note that the given dataset is divided into a training set and a test set. The training set is stored on the edge server, and the test set is stored on the IoT device. In the independent PCA framework, the IoT device first generates the extractor $\mathbf{E}_I$ by using PCA and the test set. Note that see Section 3.2 for the generation process of $\mathbf{E}_I$. Then, the IoT device uses $\mathbf{E}_I$ to extract the features of the test dataset and uploads the

TABLE 2
Accuracy of different frameworks on benchmark datasets (% ± std).

K-NN		Results							
		LEAVES	USPS	YALE	COIL20	UMIST	YALEB	ORL	COIL
$K=1$	Raw data	65.56±0.23	94.82±0.78	**64.52±0.37**	97.45±0.23	93.92±0.78	**77.44±0.37**	**92.86±0.23**	98.6±0.78
	Ind-PCA	35.37±0.23	23.39±0.78	7.63±0.37	23.63±0.23	16.81±0.78	3.52±0.37	8.63±0.23	9.6±0.78
	Ours	**67.22±0.23**	**95.15±0.78**	63.63±0.37	**98.29±0.23**	**94.03±0.78**	76.56±0.37	92.79±0.23	**99.26±0.78**
$K=3$	Raw data	72.04±0.23	94.67±0.78	**59.26**±0.37	94.34±0.23	84.73±0.78	**76.04±0.37**	81.87±0.23	97.43±0.78
	Ind-PCA	35.37±0.23	30.16±0.78	7.19±0.37	12.51±0.23	12.35±0.78	3.26±0.37	6.71±0.23	8.74±0.78
	Ours	**72.22±0.23**	**94.84±0.78**	58.89±0.37	**96.01±0.23**	**84.76±0.78**	67.4±0.37	**82.04±0.23**	**98.00±0.78**
$K=5$	Raw data	68.89±0.23	94.40±0.78	**57.04±0.37**	92.26±0.23	77.52±0.78	**76.56±0.37**	74.26±0.23	95.51±0.78
	Ind-PCA	34.26±0.23	44.01±0.78	5.93±0.37	10.47±0.23	10.78±0.78	4.85±0.37	6.58±0.23	7.86±0.78
	Ours	**70.19±0.23**	**94.51±0.78**	56.52±0.37	**94.12±0.23**	**78.13±0.78**	67.31±0.37	**75.14**±0.23	**96.37±0.78**

extracted feature data to the edge server. After receiving the uploaded feature data, the edge server performs the PCA method to obtain extractor $\mathbf{E}_S$ and uses $\mathbf{E}_S$ to extract the features from the training dataset. Note that the generation process of $\mathbf{E}_S$ is the same as that of $\mathbf{E}_I$. Finally, the features extracted by $\mathbf{E}_I$ and $\mathbf{E}_S$ are matched to obtain the result. Note that the dimension of $\mathbf{E}_S$ is the same as that of $\mathbf{E}_I$. Without prior knowledge, the dimension of $\mathbf{E}_I$ is equal to the number of positive eigenvalues corresponding to the objective function when optimizing the test set.

In our framework, the edge server first generates the extractor $\mathbf{E}$ by using PCA and the dataset and then sends the generated $\mathbf{E}$ to the IoT device. Then, the IoT device uses the received $\mathbf{E}$ to extract features from the test dataset. Finally, the IoT device uploads the extracted feature data to the edge server for feature matching, as shown in Fig. 4 (b).

To verify the extracted features, we use the K-nearest neighbor classifier and compare the accuracy of the three frameworks (*i.e.*, raw data, direct PCA framework, and our proposed framework) under $K=1$, $K=3$ and $K=5$. Note that the high accuracy indicates that the extracted features meet the needs of the application. Finally, we report the results of 50 averages. Additionally, to verify the NestE, we follow the experimental settings of the WAPL [23] and DFE [28] algorithms and show the results for setting $K=1$.

4.2 Experimental Results

4.2.1 Redundant Eigenvectors

To verify whether redundant eigenvectors are included in the extractor generated by the datasets, we divide each dataset into a training set and a test set. On the edge server, we first generate extractor $\mathbf{E}$ using PCA and dataset, and then divide the extractor $\mathbf{E}$ into r subextractors according to Theorem 1. Then, we use each subextractor to extract features from the corresponding test set, and use the nearest neighbor classifier to validate the extracted features and get the accuracy. Additionally, we also show the relationship between the eigenvalues and the number of eigenvalues, as shown in Fig. 5.

We observe that there are redundant eigenvectors on all datasets. Among them, as shown, on the LEAVES and COIL datasets, more than 60% of the vectors in the extractor $\mathbf{E}$ are redundant and can be removed directly. These redundant vectors will lead to large storage space occupation (Section 4.2.3), large network transmission traffic (Section 4.2.4), and large switching overhead (Section 4.2.5). Through these experimental results in Fig. 5, we know that the extractor generated by the PCA method contains a large number of redundant vectors. In addition, as shown in Fig. 9 and Fig. 10, we also observe similar results using different feature extraction algorithms, which motivates our research on removing redundant vectors.

4.2.2 Accuracy

Accuracy is one of the key indicators to measure the performance of the application, and it is also one of the important indicators of the effectiveness of the extracted features. Here, we use accuracy to measure the effectiveness of the extracted features. A higher accuracy indicates more effective principal component features. Table 2 shows the results of three different frameworks on eight datasets, and we make two observations.

First, it is necessary to use the application-assisted method to extract the features because the application performance of the independent framework is the worst on all datasets. As shown, on all datasets, the accuracy obtained by the independent framework is almost guesswork. On the LEAVES dataset, for the classification tasks with only three classes, the accuracy is approximately 33%. This is because the independent extraction of features of data on the IoT device may maintain the essential structure of data, but it is not conducive to the recognition task. The corresponding application-assisted method achieves the highest accuracy on most datasets (*e.g.*, LEAVES, USPS, COIL20, UMIST, COIL). The main reason is that the extractor generated in an application-assisted way can extract the features required by the application. Therefore, to obtain higher performance, it is necessary to use the assistance of the application.

Second, the features extracted by the application-assisted method obtain an accuracy comparable to that of the raw data. As shown in Table 2, the application-assisted method can achieve even higher accuracy on the LEAVES, USPS, COIL20, UMIST, and COIL datasets. In addition, the accuracies obtained by the two methods on other datasets are similar. However, using the application-assisted method to extract the features on the IoT device shows great advantages in network transmission traffic (Section 4.2.4).

4.2.3 Reduction on Storage Space

Compared with storing multiple independent subextractors, storing subextractors through parameter sharing can

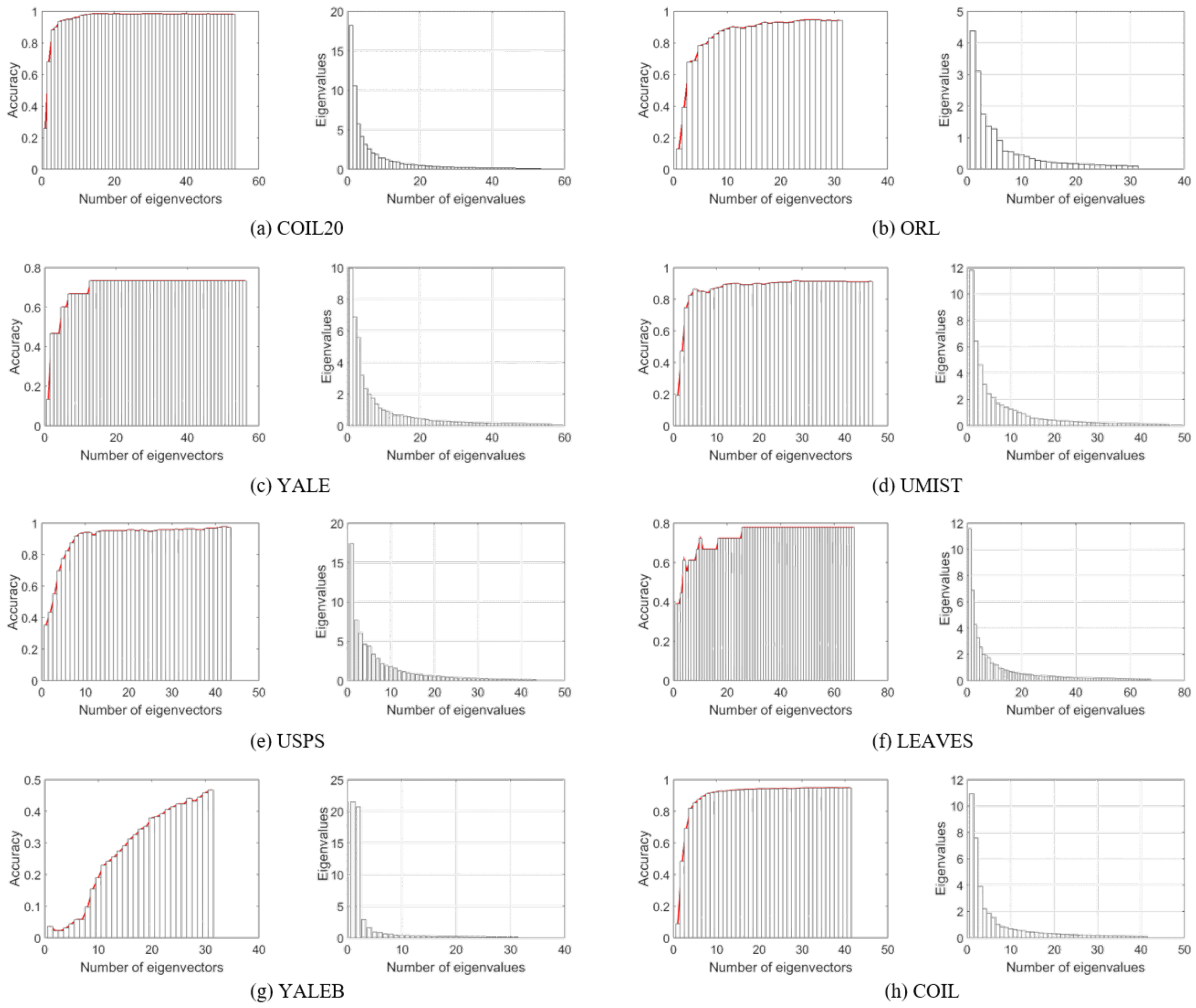


Fig. 5. Relationship between accuracy, number of eigenvectors, and eigenvalues on eight datasets by using PCA.

TABLE 3
Multiextractor and nonredundant multiextractor on storage space reduction (KB) of PCA.

Datasets	Multi-extractor	Nonredundant multiextractor	Reduced storage space
LEAVES	516	200	61.2%
USPS	83	81	2.4%
YALE	416	38.7	90.7%
COIL20	423	92.6	78.1%
UMIST	377	169	55.2%
YALEB	239	239	-
ORL	246	131	46.6%
COIL	323	100	69.0%

TABLE 4
Multiextractor and nonredundant multiextractor on storage space reduction (KB) of WAPL.

Datasets	Multi-extractor	Nonredundant multiextractor	Reduced storage space
LEAVES	338	64	81.1%
USPS	112	40	64.3%
YALE	188	116	38.3%
COIL20	254	92.6	63.5%
UMIST	92.6	46.4	49.9%
YALEB	100	84.9	15.1%
ORL	316	192	39.2%
COIL	192	61.8	67.8%

significantly reduce the storage resources occupied. This is obvious. Here, we compare the storage space occupied by the multifunctional extractor before and after the redundant subextractors are removed. Table 3 shows the results.

The nonredundant multifunctional extractor significantly reduces the storage footprint in most cases. As shown in Table 3, compared with the multifunctional extractor with redundant vectors, the generated nonredundant

TABLE 5
Multiextractor and nonredundant multiextractor on storage space reduction (KB) of DFE.

Datasets	Multi-extractor	Nonredundant multiextractor	Reduced storage space
LEAVES	632	231	63.5%
USPS	13.6	11.7	14.0%
YALE	108	92.6	14.3%
COIL20	123	54.1	56.0%
UMIST	146	131	10.2%
YALEB	293	269	8.2%
ORL	277	131	52.7%
COIL	770	246	68.1%

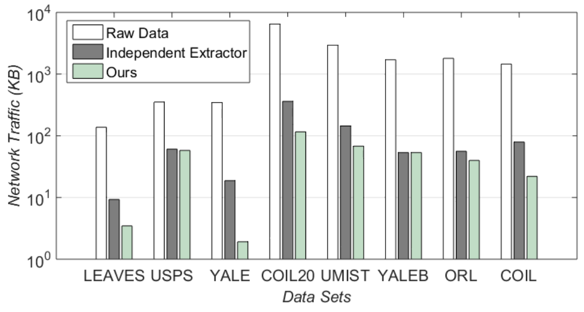


Fig. 6. Network traffic under different frameworks of PCA.

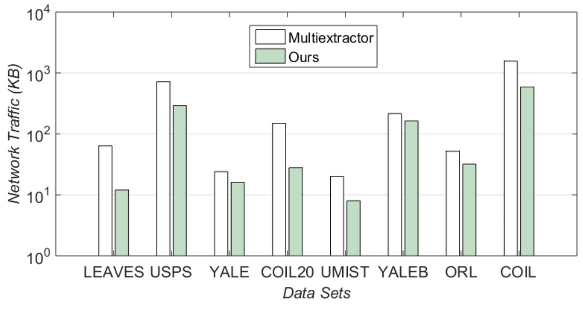


Fig. 7. Network traffic under different frameworks of WAPL.

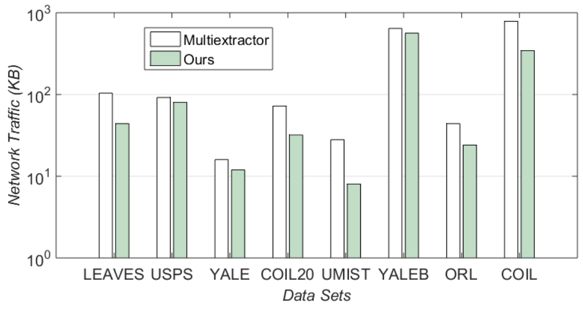


Fig. 8. Network traffic under different frameworks of DFE.

multifunctional extractor reduces the storage footprint by 61.2%, 2.4%, 90.7%, 78.1% 55.2%, 0, 46.6%, and 69.0% on the LEAVES, USPS, YALE, COIL20, UMIST, YALEB, ORL and COIL datasets, respectively. The storage space occupied by the multifunctional extractor is equal to the storage space occupied by the largest subextractor. This is because the deployment of the multifunctional extractor adopts the method in which the small-capacity subextractors are nested in the large-capacity subextractors. As shown in Fig. 5, the largest subextractor obtained from the YALEB dataset is not a redundant subextractor. Therefore, although the extractor generated on the YALEB dataset contains redundant subextractors, the storage space occupied by the remaining subextractors remains unchanged after the redundant subextractors are removed. However, in most cases, removing redundant subextractors can save considerable storage space. In addition, we also show the storage space saved by the WAPL and DFE algorithms under our proposed framework, as shown in Table 4 and Table 5. As shown, the proposed framework enables WAPL to reduce storage space by up to 81.1%, and DFE to reduce storage space by up to 68.1%. These experimental results show that removing redundant vectors is an effective way to save storage resources in IoT devices.

4.2.4 Network Traffic

The number of features extracted by the independent extractor is determined by the PCA and the test dataset. To obtain the highest accuracy, the number of features extracted by the independent extractor is equal to the number of positive eigenvalues corresponding to the eigendecomposition of its objective function. Similarly, the number of features extracted by our framework is determined by the PCA and the training dataset. Here, we compare the network traffic of our method (after removing redundant subextractors) with other methods. Fig. 6 shows the results.

Our proposed framework has minimal network traffic in all datasets. As shown in Fig. 6, on the YALE dataset, compared with not removing redundant subextractors, our proposed nonredundant multifunctional extractor reduces network traffic by 89.4%. On the COIL dataset, our proposed nonredundant multifunctional extractor reduces network traffic by 72.3%. Similar results are obtained using WAPL and DFE, as shown in Fig. 7 and Fig. 8. This is because NestE removes a large number of large redundant subextractors, avoiding the extra network traffic caused by the large-capacity redundant subextractors extracting a large number of useless features. Therefore, by removing redundant subextractors, our proposed framework leads to minimal network traffic. Note that the small-capacity subextractor can obtain the same accuracy as the large-capacity redundant subextractor.

Extracting features on IoT devices can significantly reduce the network transmission traffic that needs to be uploaded. As shown in Fig. 6, on the LEAVES dataset, compared with uploading the raw image data, uploading the extracted feature data reduces network traffic by 93.3%, and our proposed framework reduces network traffic by 97.5%. Therefore, it is important to perform feature extraction in IoT devices to reduce network transmission traffic, especially in the Internet of Everything and 5G era.

4.2.5 Reduction in Switching Overhead

The available IoT device resources are dynamically changing. To keep the application's service of data feature extraction uninterrupted, the application needs to frequently

TABLE 6
Switching overhead in memory usage for multiextractors and nonredundant multiextractors of PCA.

Datasets	Upgrade switching overhead (KB)					Downgrade switching overhead (KB)				
	PCA		NestE-PCA		Reduced overhead	PCA		NestE-PCA		Reduced overhead
	Page-in	Page-out	Page-in	Page-out		Page-in	Page-out	Page-in	Page-out	
LEAVES	508.1	0	192	0	62.2%	0	508.1	0	192	62.2%
USPS	78.9	0	75	0	4.9%	0	78.9	0	75	4.9%
YALE	408	0	31	0	92.4%	0	408	0	31	92.4%
COIL20	415	0	85	0	79.5%	0	415	0	85	79.5%
UMIST	369	0	161	0	56.4%	0	369	0	161	56.4%
YALEB	231	0	231	0	-	0	231	0	231	-
ORL	238	0	123	0	48.3%	0	238	0	123	48.3%
COIL	315	0	92	0	70.8%	0	315	0	92	70.8%

TABLE 7
Switching overhead in memory usage for multiextractors and nonredundant multiextractors of WAPL.

Datasets	Upgrade switching overhead (KB)					Downgrade switching overhead (KB)				
	WAPL		NestE-WAPL		Reduced overhead	WAPL		NestE-WAPL		Reduced overhead
	Page-in	Page-out	Page-in	Page-out		Page-in	Page-out	Page-in	Page-out	
LEAVES	330	0	56	0	83.0%	0	330	0	56	83.0%
USPS	110	0	38	0	65.5%	0	110	0	38	65.5%
YALE	180	0	108	0	40.0%	0	180	0	108	40.0%
COIL20	246	0	84.6	0	65.6%	0	246	0	84.6	65.6%
UMIST	84.6	0	38.4	0	54.6%	0	84.6	0	38.4	54.6%
YALEB	92	0	76.9	0	16.4%	0	92	0	76.9	16.4%
ORL	308	0	184	0	40.2%	0	308	0	184	40.2%
COIL	184	0	53.8	0	70.7%	0	184	0	53.8	70.7%

switch subextractors of different capacities. Here, based on the built prototype system, we measure the memory resources occupied by parameters switching from the smallest-capacity subextractor to the largest-capacity subextractor (or from the largest-capacity subextractor to the smallest-capacity subextractor) as the switching overhead. Table 6, Table 7, and Table 8 show the results. Note that we will use CPU usage or other measures of switching overhead in our future work.

Our proposed nonredundant multifunctional extractor significantly reduces the switching overhead of subextractors. Table 6, Table 7, and Table 8 show the switching overhead of upgrading and downgrading subextractors. To be fair, when upgrading the subextractors, we quantify the switching overhead when switching from the smallest-capacity subextractor to the largest-capacity subextractor. As shown in Table 6, on the YALE dataset, the extractor with redundant subextractors needs to page-in 408 KB parameters, and our proposed framework needs to page-in 31 KB parameters, which reduces the number of page-in parameters by 92.4%. On the COIL dataset, the extractor with redundant subextractors needs to page-in 315 KB parameters, and our proposed framework needs to page-in 92 KB parameters, which reduces the number of page-in parameters by 70.8%. Similar settings also occur when downgrading subextractors; that is, we quantify the switching overhead when switching from the largest-capacity subextractor to the smallest-capacity subextractor. As shown in Table 6, on the YALE dataset, our proposed framework reduces the number of page-out parameters by 92.4%. On the COIL dataset, our framework reduces the number of page-out parameters by 70.8%. In addition, Table 7 and Table 8 show similar experimental results. This is because in the nested extractor that includes redundant subextractors, the larger subextractor is a redundant subextractor, which results in larger switching overhead.

In addition, compared with deploying multiple independent extractors, deploying nested extractors can significantly reduce the switching overhead of subextractors. This is because when upgrading the subextractor, the nested extractor only needs to page-in the parameters contained in the subextractor with a larger capacity and because these parameters will continue to be used by the larger-capacity subextractor. When downgrading the subextractor, the nested extractor only needs to page-out the extra parameters that the subextractor with a smaller capacity does not

TABLE 8
Switching overhead in memory usage for multiextractors and nonredundant multiextractors of DFE.

Datasets	Upgrade switching overhead (KB)					Downgrade switching overhead (KB)				
	DFE		NestE-DFE		Reduced overhead	DFE		NestE-DFE		Reduced overhead
	Page-in	Page-out	Page-in	Page-out		Page-in	Page-out	Page-in	Page-out	
LEAVES	624	0	223	0	64.3%	0	624	0	223	64.3%
USPS	11.5	0	9.6	0	16.5%	0	11.5	0	9.6	16.5%
YALE	100	0	85	0	15.0%	0	100	0	85	15.0%
COIL20	115	0	46	0	60.0%	0	115	0	46	60.0%
UMIST	138	0	123	0	10.9%	0	138	0	123	10.9%
YALEB	285	0	261	0	8.4%	0	285	0	261	8.4%
ORL	269	0	123	0	54.3%	0	269	0	123	54.3%
COIL	762	0	238	0	86.8%	0	762	0	238	86.8%

have. Whether upgrading or downgrading subextractors, deploying multiple subextractors independently requires paging-in and paging-out entire subextractors.

5 CONCLUSION

This paper presented a holistic design of a resource-efficient feature extraction framework for IoT devices. To extract features required by the application, the proposed framework generates the extractor with the assistance of the dataset in the edge server. Subsequently, the proposed framework proposes NestE, which generates a nonredundant multifunctional extractor to adapt to IoT devices with limited resources and dynamic changes in available resources. Experimental results show that the proposed framework substantially outperforms state-of-the-art methods in terms of accuracy, memory space reduction, network traffic, and switching overhead.

In future work, we plan to continue research in two directions. The first is to extract the nonlinear features required by the application by introducing kernel functions (*e.g.*, Gaussian kernel, polynomial kernel), and analyze the composition of the generated extractor. The second is to study resource allocation when the IoT device runs multiple applications simultaneously. We plan to maximize the performance of multiple applications by quantifying the relationship between the performance gain and resource consumption of each application.

6 ACKNOWLEDGEMENTS

This work was supported by the Fundamental Research Funds for the Central Universities (2021RC272, 2021JBZ104), the China Postdoctoral Science Foundation (2021M700364, 2021M691424), and the National Natural Science Foundation of China (62202039, 61922017, U1934220).

REFERENCES

[1] https://infohub.delltechnologies.com/l/edge-to-core-and-the-internet-of-things-2/internet-of-things-and-data-placement, [Online; accessed 9-July-2022].

[2] https://www.techtarget.com/searchcio/DrivingITSuccess/4-Things-You-Need-to-Know-Now-About-Edge-Computing, [Online; accessed 9-July-2022].

[3] Y. Shi, S. Chen, and X. Xu, "Maga: A mobility-aware computation offloading decision for distributed mobile cloud computing," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 164–174, 2018.

[4] Y. Zhao, R. N. Calheiros, G. Gange, J. Bailey, and R. O. Sinnott, "Sla-based profit optimization resource scheduling for big data analytics-as-a-service platforms in cloud computing environments," *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1236–1253, 2021.

[5] X. Gao, R. Liu, and A. Kaushik, "Hierarchical multi-agent optimization for resource allocation in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 692–707, 2021.

[6] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.

[7] Q. Zhang, L. Gui, F. Hou, J. Chen, S. Zhu, and F. Tian, "Dynamic task offloading and resource allocation for mobile-edge computing in dense cloud ran," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3282–3299, 2020.

[8] S. Yang, F. Li, M. Shen, X. Chen, X. Fu, and Y. Wang, "Cloudlet placement and task allocation in mobile edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5853–5863, 2019.

[9] X. Wang, Z. Ning, S. Guo, and L. Wang, "Imitation learning enabled task scheduling for online vehicular edge computing," *IEEE Transactions on Mobile Computing*, vol. 21, no. 2, pp. 598–611, 2022.

[10] Z. Ning, P. Dong, X. Wang, S. Wang, X. Hu, S. Guo, T. Qie, B. Hu, and R. Y. K. Kwok, "Distributed and dynamic service placement in pervasive edge computing networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 6, pp. 1277–1292, 2021.

[11] X. Wang, Z. Ning, and S. Guo, "Multi-agent imitation learning for pervasive edge computing: A decentralized computation offloading algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 411–425, 2021.

[12] P. Hu, H. Ning, T. Qiu, Y. Zhang, and X. Luo, "Fog computing based face identification and resolution scheme in internet of things," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1910–1920, 2017.

[13] J. Li, Z. Peng, B. Xiao, and Y. Hua, "Make smartphones last a day: Pre-processing based computer vision application offloading," in *Proceedings of the IEEE International Conference on Sensing, Communication, and Networking*, 2015, pp. 462–470.

[14] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 2017, pp. 276–286.

[15] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. B. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *Proceedings of*

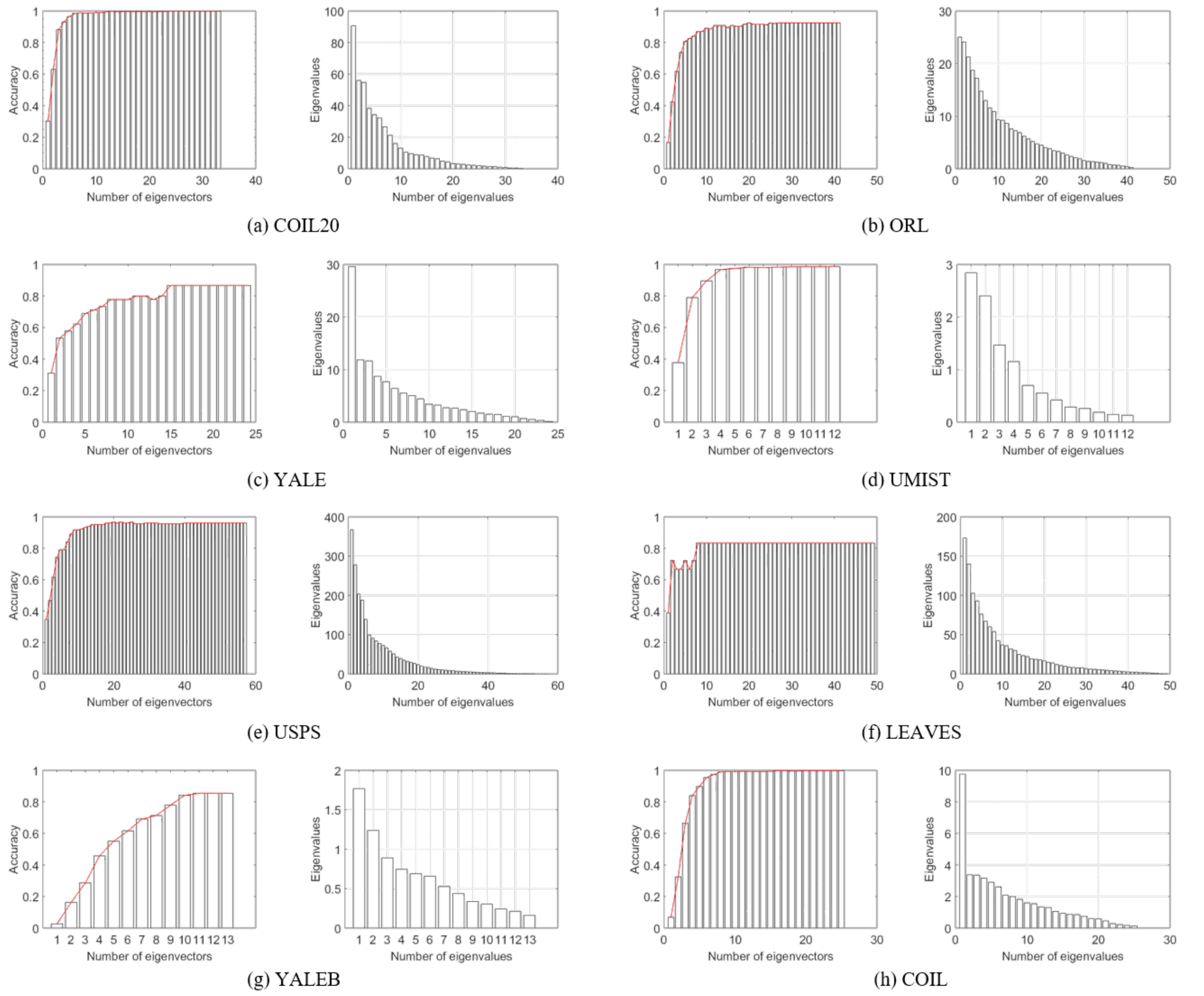


Fig. 9. Relationship between accuracy, number of eigenvectors, and eigenvalues on eight datasets by using WAPL.

the IEEE Symposium on Computers and Communications, 2012, pp. 59–66.

[16] C. Liu, Y. Cao, Y. Luo, G. Chen, V. Vokkarane, M. Yunsheng, S. Chen, and P. Hou, "A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 249–261, 2018.

[17] G. Zhao and M. Pietikainen, "Dynamic texture recognition using local binary patterns with an application to facial expressions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 915–928, 2007.

[18] X. Yi and M. G. Eramian, "Lbp-based segmentation of defocus blur," *IEEE Transactions on Image Processing*, vol. 25, no. 4, pp. 1626–1638, 2016.

[19] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[20] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[21] A. M. Martinez and A. C. Kak, "Pca versus lda," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 2, pp. 228–233, 2001.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the Neural Information Processing Systems*, 2012, pp. 1106–1114.

[23] S. Wang, C. Ding, N. Zhang, X. Liu, A. Zhou, J. Cao, and X. Shen, "A cloud-guided feature extraction approach for image retrieval in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 292–305, 2021.

[24] S. Yan, D. Xu, B. Zhang, H. jiang Zhang, Q. Yang, and S. Lin, "Graph embedding and extensions: A general framework for dimensionality reduction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 1, pp. 40–51, 2007.

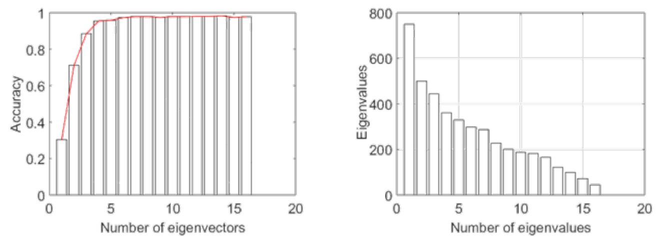
[25] C. Ding and L. Zhang, "Double adjacency graphs-based discriminant neighborhood embedding," *Pattern Recognition*, vol. 48, no. 5, pp. 1734–1742, 2015.

[26] Q. Gao, J. Liu, H. Zhang, X. Gao, and K. Li, "Joint global and local structure discriminant analysis," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 4, pp. 626–635, 2013.

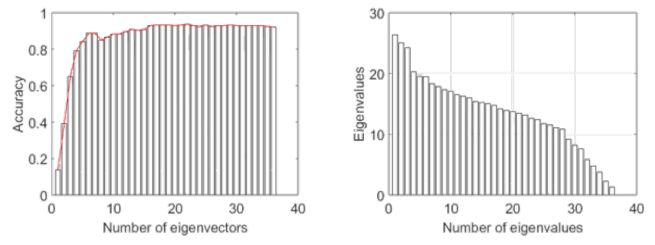
[27] X. Li, M. Chen, F. Nie, and Q. Wang, "Locality adaptive discriminant analysis," in *Proceedings of the 26-th International Joint Conference on Artificial Intelligence*, 2017, pp. 2201–2207.

[28] C. Ding, A. Zhou, X. Liu, X. Ma, and S. Wang, "Resource-aware feature extraction in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 21, no. 1, pp. 321–331, 2021.

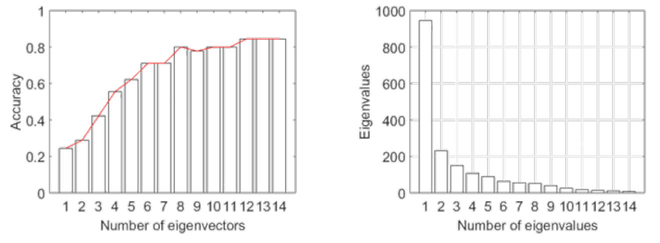
[29] X. Zhang, M. Qiao, L. Liu, Y. Xu, and W. Shi, "Collaborative cloud-



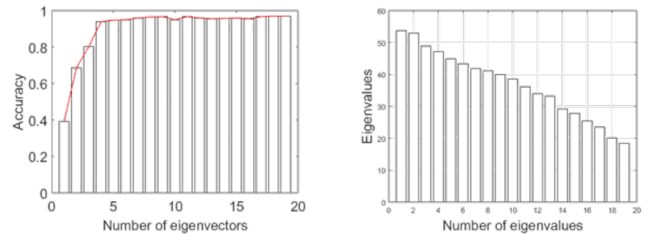
(a) COIL20



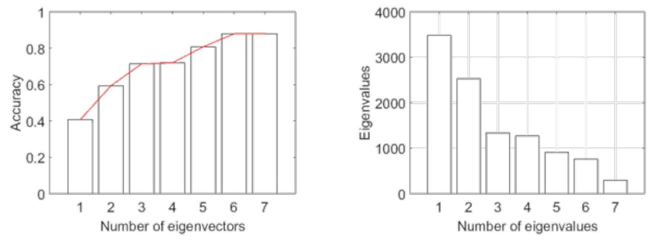
(b) ORL



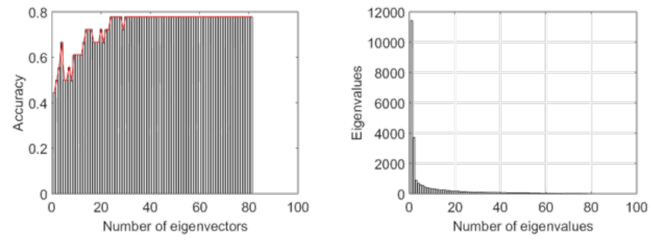
(c) YALE



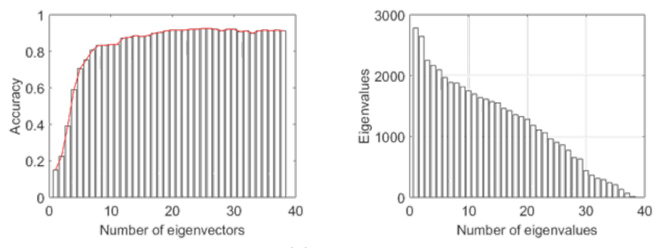
(d) UMIST



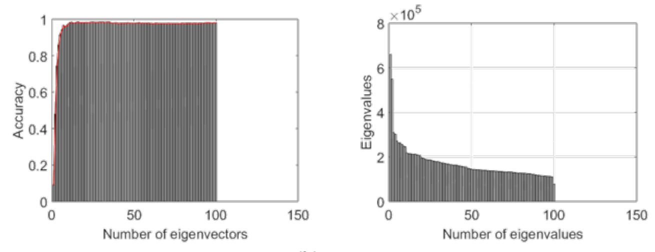
(e) USPS



(f) LEAVES



(g) YALEB



(h) COIL

Fig. 10. Relationship between accuracy, number of eigenvectors, and eigenvalues on eight datasets by using DFE.

edge computation for personalized driving behavior modeling," in *Proceedings of 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 209–221.

[30] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "Spinn: Synergistic progressive inference of neural networks over device and cloud," in *Proceedings of 16th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–15.

[31] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proceedings of 37th IEEE International Conference on Distributed Computing Systems*, 2017, pp. 328–339.

[32] J. Ren, Y. Guo, D. Zhang, Q. Liu, and Y. Zhang, "Distributed and efficient object detection in edge computing: Challenges and solutions," *IEEE Network*, vol. 32, no. 6, pp. 137–143, 2018.

[33] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 615–629.

[34] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *Proceedings of IEEE Conference on Computer Communications*, 2019, pp. 1423–1431.

[35] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *Proceedings of IEEE 24th International Conference on Parallel and Distributed Systems*, 2018, pp. 671–678.

[36] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems*, 2016, pp. 123–136.

[37] J. Wang, X. Zhu, W. Bao, and L. Liu, "A utility-aware approach to redundant data upload in cooperative mobile cloud," in *Proceedings of the International Conference on Cloud Computing*, 2016, pp. 384–391.

[38] H. Yan, X. Zhang, H. Chen, Y. Zhou, W. Bao, and L. T. Yang, "Deed: Dynamic energy-efficient data offloading for iot applications under unstable channel conditions," *Future Generation Computer Systems*, vol. 96, pp. 425–437, 2019.

[39] H. Yan, Y. Li, X. Zhu, D. Zhang, J. Wang, H. Chen, and W. Bao, "Ease: Energy-efficient task scheduling for edge computing under uncertain runtime and unstable communication conditions," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 7, pp. 425–437, 2021.

[40] A. Breloy, S. Kumar, Y. Sun, and D. P. Palomar, "Majorization-minimization on the stiefel manifold with application to robust sparse pca," *IEEE Transactions on Signal Processing*, vol. 69, pp. 1507–1520, 2021.

[41] F. Shang, J. Cheng, Y. Liu, Z.-Q. Luo, and Z. Lin, “Bilinear factor matrix norm minimization for robust pca: Algorithms and applications,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 9, pp. 2066–2080, 2018.

[42] N. Xue, J. Deng, S. Cheng, Y. Panagakis, and S. Zafeiriou, “Side information for face completion: A robust pca approach,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 10, pp. 2349–2364, 2019.

[43] C. Kim and D. Klabjan, “A simple and fast algorithm for l1-norm kernel pca,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 8, pp. 1842–1855, 2020.

[44] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[45] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2261–2269.

[46] G. H. Golub and C. F. V. Loan, *Matrix Computations*. Baltimore, USA: Johns Hopkins University Press, 1996.

[47] J. Duchene and S. Leclercq, “An optimal transformation for discriminant and principal component analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 6, pp. 978–983, 1988.

[48] “Handwritten Digits USPS data set,” https://cs.nyu.edu/ roweis/data.html, [Online; accessed 9-March-2022].

[49] “Yale Face Database,” http://vision.ucsd.edu/content/yale-face-database, [Online; accessed 9-March-2022].

[50] “Columbia University Image Library (COIL-20),” https://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php, [Online; accessed 5-June-2022].

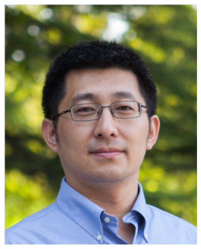
[51] “UMIST Database,” https://cs.nyu.edu/ roweis/data.html, [Online; accessed 5-June-2022].

[52] “The Extended Yale Face Database B,” http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html, [Online; accessed 9-March-2022].

[53] “The ORL Database of Faces,” https://cam-orl.co.uk/facedatabase.html, [Online; accessed 9-March-2022].

[54] “Columbia Object Image Library Dataset(COIL-100),” https://www.cs.columbia.edu/CAVE/software/softlib/coil-100.php, [Online; accessed 9-March-2022].

Chuntao Ding is currently a lecturer at Beijing Jiaotong University. He received his Ph.D. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2021. He also worked as a research assistant at Hong Kong Polytechnic University and as a visiting scholar at Michigan State University. His research interests include Edge Computing and Machine Learning.



Yidong Li is a professor in the School of Computer and Information Technology at Beijing Jiaotong University. Dr. Li received his B.Eng. degree in electrical and electronic engineering from Beijing Jiaotong University in 2003, and M.Sci. and Ph.D. degrees in computer science from the University of Adelaide, in 2006 and 2010, respectively. Dr. Li’s research interests include big data analysis, privacy preserving and information security and intelligent transportation. Dr. Li has published more than 150 research papers in various journals and refereed conferences. He has organized several international conferences and workshops and has also served as a program committee member for several major international conferences.

Zhichao Lu received his Ph.D. degree in Electrical and Computer Engineering from Michigan State University, USA, in 2020. He is currently a post-doc research fellow at Southern University of Science and Technology, China. His research interests include machine learning assisted evolutionary algorithms, automated machine learning, and in particular evolutionary neural architecture search. He received the GECCO-2019 best paper award.

Shangguang Wang is a professor at the School of Computer Science, Beijing University of Posts and Telecommunications, China. He is the founder&chief scientist of Tiansuan Constellation. He is also the director of Star Network and Intelligence Computing Laboratory, and vice director of Sate Key Laboratory of Networking and Switching at BUPT. His research interests include service computing, mobile edge computing, cloud computing, and satellite computing. He is currently serving as chair of IEEE Technical Committy on Services Computing(TCSVC). He also served as general chairs or program chairs of 10+ IEEE conferences, advisor/associate editors of several journals.

Song Guo received the PhD degree in computer science from the University of Ottawa, and was a professor with the University of Aizu. He is a full professor with the Department of Computing, The Hong Kong Polytechnic University. His research interests include the areas of Big Data, Cloud Computing and networking, and distributed systems. He is the recipient of the 2017 IEEE Systems Journal Annual Best Paper Award and other five Best Paper Awards from IEEE/ACM conferences. He was an associate editor of the IEEE Transactions on Parallel and Distributed Systems and an IEEE ComSoc Distinguished Lecturer. He is now an the editorial board of the IEEE Transactions on Emerging Topics in Computing, the IEEE Transactions on Sustainable Computing, the IEEE Transactions on Green Communications and Networking, and the IEEE Communications. His currently serves as a director and member of the Board of Governors of ComSoc.