# Efficient Privacy Preserving Edge Intelligent Computing Framework for Image Classification in IoT

Omobayode Fagbohungbe, *Student Member, IEEE*, Sheikh Rufsan Reza, *Student Member, IEEE*, Xishuang Dong, *Member, IEEE*, and Lijun Qian, *Senior Member, IEEE*

***Abstract*—To extract knowledge from the large data collected by edge devices, a traditional cloud-based approach that requires data upload may not be feasible due to communication bandwidth limitations as well as privacy and security concerns of end-users. A novel privacy-preserving edge intelligent computing framework for image classification in IoT is proposed to address these challenges. Specifically, the autoencoder will be trained unsupervised at each edge device individually, and then the obtained latent vectors transmitted to the edge server for the training of a classifier. This framework would reduce the communication overhead and protect end-users' data. Compared to federated learning, the training of the classifier in the proposed framework is not subject to the constraints of the edge devices, and the autoencoder can be trained independently at each edge device without any server involvement. Compared to collaborative intelligence such as SplitNN, the proposed method does not suffer from high communication cost as noticed in SplitNN. Furthermore, the privacy of the end-users' data is protected by transmitting latent vectors and without the additional cost of encryption. Experimental results provide insights on the image classification performance vs. various design parameters such as the data compression ratio of the autoencoder and the model complexity.**

***Index Terms*—Deep learning, edge computing, autoencoder, convolutional neural network, internet of things, privacy preserving deep learning.**

## I. Introduction and Motivation

EMERGING Technologies such as the Internet of Things (IoT) and 5 G networks will add a huge number of devices and new services. As a result, a huge amount of data will be generated in real-time. One of the important data types is image data since many applications such as video surveillance, autonomous driving, etc., involve images and videos. To take advantage of the "big image data," data analytics must be performed to extract knowledge from the data. One way to handle the data would be by uploading all the data from edge devices to the cloud or remote data centers for processing and knowledge extraction [1]. However, as highlighted in Fig. 1, several factors may render this practice infeasible: 1) The sheer volume of the images may overwhelm an uplink with limited bandwidth; 2) The uplink may not always be available, e.g., when wireless communication is used, there might be downtime due to weather(in the case of mmWave), distance, or jamming; 3) Proprietary images may need encryption which introduces additional delay; 4) The end users may have concerns about the security and privacy of their images; thus they may not agree to upload raw images that may contain private information. Furthermore, uploading is subject to eavesdropping, interceptions, or other unauthorized access.

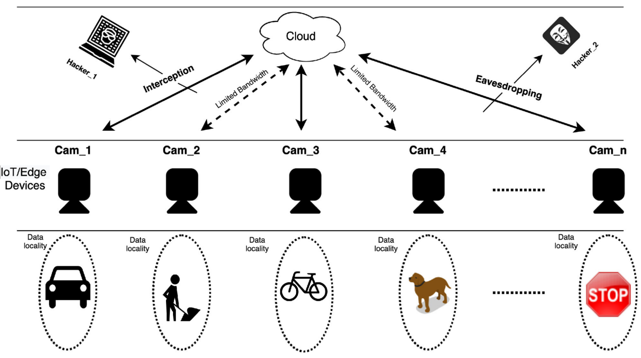


Fig. 1. Challenges incurred when uploading all data from edge devices to the cloud.

A novel efficient privacy-preserving framework for image classification in edge intelligent computing systems is proposed in this paper to address these challenges. Specifically, the large raw data will be processed locally (at the edge) by a pre-trained autoencoder. And instead of uploading the raw image, only the compressed latent vectors that contain critical features learned from the raw image will be uploaded through the access point or hub to the edge server for further processing. The proposed framework is highlighted in Fig. 2. The experiments demonstrate that the learning performance of extracting knowledge at the server has minimal degradation when the compression ratio is not significant (e.g., below 16 in our test cases). Furthermore, the raw images can be reconstructed with minimal error at the server using the pre-trained decoder if available and needed. The proposed framework encourages the design and implementation

Manuscript received 16 February 2021; revised 7 June 2021 and 15 July 2021; accepted 6 August 2021. Date of publication 15 October 2021; date of current version 22 July 2022. This work was supported in part by U.S. National Science Foundation (NSF) Award under Grant 2018945 and in part by the U.S. Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)) under Agreement number FA8750-15-2-0119. *(Corresponding author: Omobayode Fagbohungbe.)*

The authors are with the CREDIT Center and the Department of Electrical and Computer Engineering, Prairie View A&M University, Texas A&M University System, Prairie View, TX 77446 USA (e-mail: ofagbohungbe@pvamu.edu; sreza@student.pvamu.edu; xidong@pvamu.edu; liqian@pvamu.edu).

Digital Object Identifier 10.1109/TETCI.2021.3111636

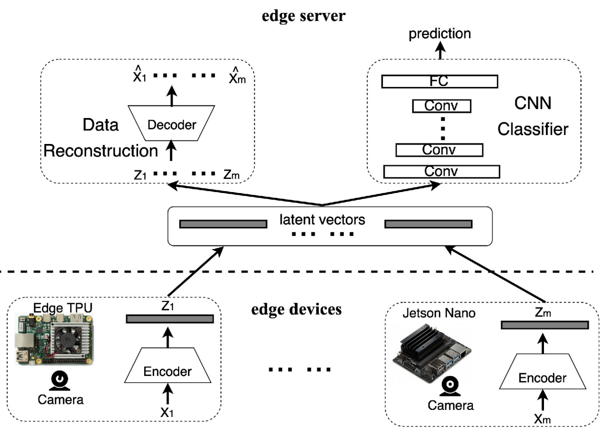


Fig. 2. The proposed efficient privacy preserving framework for image classification in edge computing systems. Here $x_i$ is the raw image, $z_i$ is the compressed latent vector, and $\hat{x}_i$ is the reconstructed image.

of emerging edge intelligent computing that are both efficient and privacy preserving for 5 G, IoT, and other advanced technologies.

Compared to traditional source coding (e.g., zip), using an autoencoder has the following advantages: 1) Instead of only reducing the redundancy in the raw data as in source coding or traditional data compression, an autoencoder will extract critical features in the raw data and encode the features in a compact form (the latent vector), in other words, the encoder performs initial learning at the edge devices; 2) In addition to compressing the data, the autoencoder also "encrypts" the data by transforming the raw data into latent vectors, thereby enhancing the security of data. For example, a zipped file can easily be unzipped by an adversary if not encrypted; on the contrary, an adversary can not reconstruct the raw data from the latent vector without knowing the structure (e.g., number of layers, number of nodes in each layer) and all the weights of the pre-trained autoencoder (specifically the decoder part). It is shown in [2] that the autoencoder provides a similar level of security to standard encryption - assuming that the decoder is not shared; (3) Even if an adversary captures the edge device, it is very challenging for the adversary to deduce the decoder part from the encoder part on the edge device.

The proposed framework has some similar characteristics such as taking advantage of large and diverse data from many edge devices and data locality at each device as in federated learning [3]–[6] and collaborative intelligence such as SplitNN [7]. However, compared to federated learning and collaborative intelligence, the proposed framework has the following advantages:

1) In federated learning, the server and the end-users (edge devices) train the same model. As a result, the model's complexity is constrained by the edge device's computing capability and storage capacity. On the contrary, *in the proposed framework, the training of the classifier is done at the edge server only. Thus, it can be deep and complex if needed, and it is not subject to the constraints of the edge devices.*
2) In federated learning, the edge device must rely on the server to update the gradients and train the model. *In the proposed framework, the training of the autoencoder can be done independently at each edge device without any server involvement*.
3) In federated learning, the privacy of the end-users' data is protected by applying differential privacy schemes [8] or through secure aggregation [9], thus introduce additional cost due to encryption or secret sharing. *In the proposed framework, the privacy of the end-users' data is protected by transmitting latent vectors without the additional cost of encryption*.
4) In collaborative intelligence, the volume of the intermediate feature tensor at the early layers or optimal layer might be larger than that of the original input, and so uploading large amount of intermediate features to the server can lead to higher transmission latency and energy consumption [10], [11]. *However, the proposed framework will always compress the original data to latent vectors with the compression ratio controlled to provide additional flexibility*. Although it is possible to compress the intermediate features before sending them to the server in collaborative intelligence, this implies an additional compression step is introduced. On the contrary, *the proposed autoencoder will extract salient features as latent vectors and perform controlled compression at the same time without the need for an additional step for compression.*
5) In SplitNN, the training of the model is done sequentially. As a result, the training may be very time consuming when the number of edge devices is significant. On the contrary, the autoencoders at the edge devices can be trained in parallel in the proposed framework. Also, constant communication between the edge device and server is required to exchange intermediate features and gradients in SplitNN, which incurs much overhead. *In the proposed framework, the edge device will send the latent vectors only once for the server to train the CNN classifier. This mode of communication leads to low communication cost and overhead reductions.*

The proposed framework is explained in Section II. Section III gives the design and implementation details. Experimental results and analysis are given in Section IV. Further discussions and related works are reviewed in Section V. Section VI concludes the paper.

## II. Proposed Framework

The proposed efficient privacy-preserving framework for image classification in edge intelligent computing systems is shown in Fig. 2. It has two levels: the edge devices and the edge server. It is assumed that the nodes of the edge devices contain sensors such as cameras and embedded computing devices such as Google edge TPU [12] or NVIDIA Jetson Nano [13]. The edge server is assumed to have large storage and strong computational capacity. The edge segment of the framework mainly contains the various edge devices of interest and the pre-trained encoder. The server mainly contains the hub, the pre-trained classifier, and

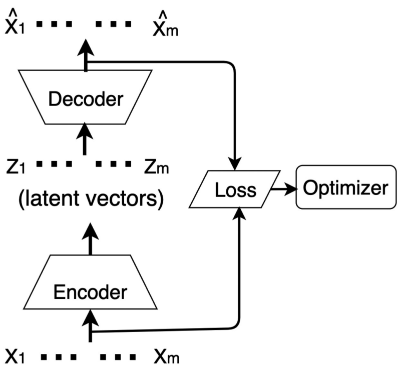


Fig. 3. The training for the proposed autoencoder at edge device.

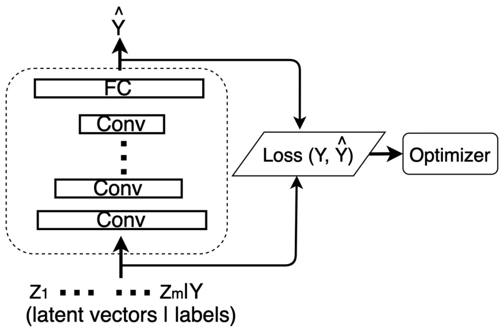


Fig. 4. The training for the proposed CNN classifier at the server.

the pre-trained decoder. We only consider supervised learning in this paper, and it is assumed that the training dataset is labeled. One of the motivations for this proposed framework stems from the guaranteed reduction in feature size of the original input when observed at the encoder output. With a feature (latent vector) size smaller than the size of the original input being sent to the server, the latency and the energy overhead and communication cost can be reduced. Furthermore, this provides improved data privacy and security, compared to sending the raw data to the server.

The data from each edge device is passed to the corresponding encoder attached to it. A unique pre-trained encoder is used at each edge device to take advantage of data locality at each device. The function of the pre-trained encoder in the inference mode is to extract the most important and critical features in the data. The encoder also ensures dimension reduction of the input data by a pre-determined factor. The extracted critical features (latent vectors, intermediate features, or feature maps when the data are images) are then transmitted to the hub at the server. The two primary tasks at the server are the classification task and the data reconstruction task (recover a copy of the original image from the latent vectors). In other words, at the server, the latent vectors are input to the pre-trained classifier for prediction and are also input to the corresponding decoder for the reconstruction of the images.

The design of the proposed framework has two (2) stages: the training stage and the testing stage.

## *A. Training Stage*

The dataset collected at each edge device is used to train an autoencoder for the corresponding device as a way to take advantage of the data locality at each device. Autoencoders are generative models where an artificial neural network is trained to reconstruct its input in an unsupervised way. Fig. 3 illustrates all the components of an autoencoder and the training process. It is made up of two main blocks, which are the encoder and the decoder [14], [15]. The encoder compresses the input $X$ into a low dimensional representation of pre-determined size, called the latent vector denoted by $Z$ that contains the most important features in the data. When the input data are images, $Z$ will be the corresponding feature maps. The mapping function of an encoder is stated in equation 1 where $Z$ is the encoder output, $W$ is the model weight and $b_e$ is the bias of the encoder, $X$ is the model input and $f(.)$ is the non-linear activation function.

$$Z = f_\theta(X) = f(WX + b_e) \tag{1}$$

The decoder then tries to reconstruct the original input data/image from the latent vector $Z$. The reconstructed input data obtained at the decoder output is denoted by $\hat{X}$. It should be noted that an autoencoder is a lossy network as the original image will not be fully recovered. However, it is expected that the critical features will remain in the recovered image. The decoder is represented mathematically in equation 2 where $\hat{X}$ is the decoder output or estimated input, $V$ is the decoder weight, $Z$ is the encoder output, $b_d$ is the decoder bias and $g(.)$ is the activation function of the decoder.

$$\hat{X} = g_{\theta'}(Z) = g(VZ + b_d) \tag{2}$$

The autoencoder achieves the proper training of the encoder and decoder by minimizing the differences between the original input ($X$) and the reconstructed input ($\hat{X}$). The training is achieved using the mean square error (MSE) loss function or any other appropriate loss function. The formulae for the MSE loss function is stated in equation 3. After the training of the autoencoder, the encoder part of the autoencoder is then extracted, deployed in the inference mode on the edge device, and then used to generate the latent vector $Z$. Hence, the dataset is transformed from $[X, Y]$ to $[Z, Y]$ where $Y$ are the labels.

$$L_{\theta,\theta'} = \frac{\sum_{i=1}^{N} ||X_i - \hat{X}_i||_2^2}{N} \tag{3}$$

The latent vectors and the corresponding labels are aggregated at the hub, and then used to train a classifier on the cloud in a supervised manner, as shown in Fig. 4. The type of classifier at the cloud is determined by the type of supervised task to be done. The most common type of classifier used for image dataset is the convolutional neural network (CNN) and it is used as the classifier in this work. CNN is a type of multilayer neural network that preserves spatial relationships by performing convolution operation in order to learn features at different layers. The mathematical equation for a convolution operation is given in (4) where $S(i, j)$ is called the feature map, $K(i, j)$ is the filter, and $X(m, n)$ is the input.The convolution operation, which is equivalent to an integral that expresses the amount of overlap of $K$ as it is shifted over $X$, is achieved by taking the dot product

**Algorithm 1:** Autoencoder model development for image dataset at each edge device.

```
Input: Training Image data at each edge device $X$. The
   corresponding labels is also $X$
   Split dataset into training image dataset(70%) and
   testing image dataset (30%)
   Normalize the data: $X = \frac{x-min(x)}{max(x)-min(x)}$
Train the autoencoder model:
 1:  initialize $\boldsymbol{\theta}$
     Training Process
 2:  for $numberofepochs$ do
 3:     Autoencoder forward pass $\longrightarrow \hat{X}_i$
 4:     calculate loss function $L \longrightarrow |X - \hat{X}_i|$
 5:     perform back propagation $\longrightarrow \frac{\partial L}{\partial \theta_i}$
 6:     update autoencoder weights, $\theta_{i+1} \longrightarrow \theta_i$ - $\eta \frac{\partial L}{\partial \theta_i}$
 7:  end for
 8:  return $\boldsymbol{\theta}$
Test the autoencoder model: Testing Process
 9:  for $XinTestingImagedataset$ do
10:     autoencoder forward pass $\longrightarrow \hat{X}_i$
11:     encoder forward pass $\longrightarrow Z_i$
12:     Loss $\longrightarrow \frac{\sum_{i=1}^{N} ||X_i - \hat{X}_i||}{N}$
13:  end for
14:  return $Loss$, $Z$
```

**Algorithm 2:** CNN Model development for latent variables at the edge server.

```
Input: The compressed/machine intelligible imaga dataset
   $Z$ and corresponding labels $Y$ at the cloud
   Split dataset into training (70%) and testing (30%)
   Normalize the data: $z = \frac{z-min(z)}{max(z)-min(z)}$
Train the autoencoder model:
 1:  initialize $\boldsymbol{\theta}$
     Training Process
 2:  for $numberofepochs$ do
 3:     CNN forward pass $\longrightarrow \hat{Y}_i$
 4:     calculate loss function $L \longrightarrow -\sum_{i=1}^{M} Y_i \log(\hat{Y}_i)$
 5:     perform back propagation $\longrightarrow \frac{\partial L}{\partial \theta_i}$
 6:     update CNN weights, $\theta_{i+1} \longrightarrow \theta_i$ - $\eta \frac{\partial L}{\partial \theta_i}$
 7:  end for
 8:  return $\boldsymbol{\theta}$
Test the CNN model: Testing Process
 9:  for $Z$ in $TestingImagedataset$ do
10:     CNN forward pass $\longrightarrow \hat{Y}_i$
11:     Accuracy $\longrightarrow \frac{\sum_{i=1}^{N} 1(Y_i == \hat{Y}_i)}{\sum_{i=1}^{N} Y_i}$
12:  end for
13:  return $Accuracy$
```

of two inputs over a finite number of samples. [15], [16].

$$S(i,j) = (K * X)(i,j) = \sum_m \sum_n X(i-m, j-n) K(m,n) \tag{4}$$

The mathematical representation of forward propagation for a feature map at a particular layer is given by (5) where $S_j^l$ is the $j$-th feature map in $l$-th layer, $S_j^{l-1} (m = 1, \ldots, M)$ are the outputs of the $l$-1th layer, $w_{jm}^l$ is the weight connected to the $m$-th feature map in the previous layer, $b_j^l$ is the $j$-th bias of the $l$-th layer, and $F$ is the activation function [17]. The cross entropy loss function which results in normalized probabilities is used for training the classifier at the edge server. The mathematical representation of the cross entropy loss is shown in 6 where $Y_i$ is the label/ground true and $\hat{Y}_i (0 \geq Y_i \geq 1)$ is the prediction probabilities.

$$S_j^l = F\left( \sum_{m=1}^{M} w_{jm}^l * S_m^{l-1} + b_j^l \right) \tag{5}$$

$$L(Y_i, \hat{Y}_i) = -\sum_{i=1}^{M} Y_i \log(\hat{Y}_i) \tag{6}$$

The algorithm for the training phase is stated in Algorithm 1 and Algorithm 2.

## B. Inference Stage

In this stage, the pre-trained encoder, pre-trained decoder, and pre-trained classifier are deployed in the inference mode. The data $X$ from a edge device is fed to the corresponding pre-trained encoder attached to that device. The encoder then transforms the data $X$ to a latent vector $Z$, representing the most critical feature in $X$. The latent vector $Z$, which is smaller than $X$ by a pre-determined ratio, is then transmitted to the edge server. The latent vector $Z$ is fed into the pre-trained classifier at the edge server, and the classifier then predicts a label $\hat{Y}$. The original data is sometimes needed at the edge server in applications such as anomaly detection and security surveillance. When a copy of the original image is needed at the server, the latent vector $Z$ is fed into the input of the corresponding decoder, and the estimate of the original data is obtained. In applications where privacy is very important, the original image might not be requested due to privacy concerns.

## C. Proposed Framework and Data Streaming

The design of the deep learning framework above is done using an offline learning approach. In offline training, historical data is available at the edge to train the autoencoder during the training phase. In IoT data streaming, the proposed framework still applies, particularly in a situation where the streamed data needs to be stored for labeling to take place. In situations where this is not possible, the proposed framework can still be used with some slight modification depending on the velocity of the data. Firstly, the online learning approach is used as the datasets are not available at once. This means that the traditional backpropagation method used above might change to a backpropagation method that is suitable for online learning, such as hedge backpropagation [18] or use the traditional backpropagation with a batch size of one which is inefficient [19]. Furthermore, there will be a need for a distributed streaming processing platform such as Apache Flink, Apache Spark, Kafka streams to handle

TABLE I
INFORMATION ON THE CIFAR10 AND IMAGENET (IMGNETA AND IMGNETB) DATASETS

| Dataset | Image size | # of images | Training Testing ratio | # of classes | Comments |
|---|---|---|---|---|---|
| CIFAR10 | 32*32*3 | 60 000 | 5:1 | 10 | |
| IMGNET-A | 224*224*3 | 13,000 | 7:3 | 10 | very different images |
| IMGNET-B | 224*224*3 | 13,000 | 7:3 | 10 | very similar images |

issues peculiar to data streaming such as out-of-order datasets and data buffering in order to balance event-processing with low latency and high throughput [20].

## D. Security Analysis

The security of the proposed framework, which is judged by how difficult the original image can be recovered from the transmitted compressed image, is analyzed in this section. Assuming the compressed image is intercepted during transmission, it is possible to recover the original image if the pre-trained weights and other parameters of the decoder associated with the intermediate features are known. This method is impossible as the parameters of the decoder are not known as they are not transmitted. The original image can still be recovered by building a model using a dataset of the input image and the corresponding intermediate features as stated in [21], [22]. However, for our proposed framework, this method is impossible as the input image is not available or transmitted. Furthermore, another possible method to recover the original image is by training a decoder using the pre-trained weights of the encoder and other parameters of the autoencoder used in generating the intermediate features. However, the pre-trained weights of the encoder and other parameters of the autoencoder are not transmitted to the server (only the intermediate feature is transmitted) or known, making this method impossible or very challenging. This method also requires the input original input which is not available. This recovery is a very non-trivial problem as there are infinitely large possible model configurations to train and to check if they can reconstruct the original image. The mathematical proof to show that it is challenging to reconstruct the input image from the compressed image is carried out in [7].

## III. EXPERIMENTS

## A. Dataset Description

The result in this work is generated using three different datasets summarized in Table I. These datasets are from the Canadian Institute For Advanced Research dataset (CIFAR10) [23] and the ILSVRC (ImageNet) 2012 datasets [24].

*1) Canadian Institute for Advanced Research (CIFAR10):* This dataset containing 60,000 color images is a subset of about 80 million labeled but tiny images. The dataset is further divided into 50,000 training samples and 10,000 testing samples, each of dimension $32 \times 32 \times 3$. It has ten (10) mutually exclusive classes with no semantic overlaps between images from different classes.

*2) ILSVRC (ImageNet) 2012:* The original ILSVRC 2012 dataset contains about 1.2 million color images of different sizes across about 1,000 classes. The 1,000 classes are either internal

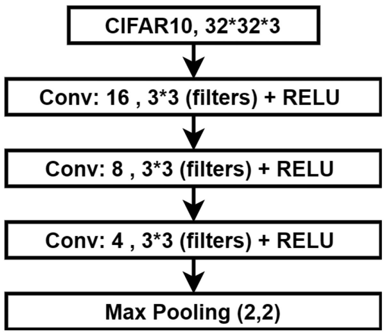

CIFAR10, 32*32*3
Conv: 16 , 3*3 (filters) + RELU
Conv: 8 , 3*3 (filters) + RELU
Conv: 4 , 3*3 (filters) + RELU
Max Pooling (2,2)

Fig. 5. Details of an encoder model for compression size of 4 using CIFAR10 dataset.

or leaf nodes but they do not overlap. Two subsets of the ILSVRC 2012 dataset termed IMGNET-A and IMGNET-B are used in this work. Each subset contains about 13,000 images each resized to a dimension $224 \times 224 \times 3$, spanning 10 classes. Each subset dataset is further divided into training samples and testing samples with a ratio of 7:3. The difference between the two subsets lies in the type of nodes they contain. The IMGNET-A subset contains images from 10 different leaf nodes (diverse images), while IMGNET-B contains ten (10) child nodes from a single leaf node (similar images).

## B. Deep Learning Model Design and Training Strategy

The autoencoder for the edge devices and the classifier at the edge server are chosen because the autoencoder is optimized for feature extraction and the classifier is optimized for image classification.

*1) Autoencoder Design and Training Strategy:* The autoencoder architecture is affected by the type of images and the compression ratio. For instance, the model architecture for the CIFAR10 dataset for compression ratios 4 and 8 are different. This condition also applies to compression ratio 4 for IMGNET-A and CIFAR10 datasets. Hence, different models are developed across several edge devices, compression ratio, and datasets.

Fig. 5 shows the model architecture for an encoder designed for the CIFAR10 dataset for a compression ratio of 4. In general, the autoencoder model contains a mix of convolutional (same padding), max pooling, and upsampling layers. The ReLu function is used as the activation function for all layers except the last layer, where the sigmoid function is used.

In this work, the autoencoder models are trained from scratch using the glorot-uniform method as the initializer, mean square error as the cost function, and rmsprop optimization algorithm as the optimizer. After the convergence of the autoencoder model during the training process, the encoder part of the autoencoder is then extracted, and deployed in the inference mode to compress

TABLE II
THE DEEP LEARNING MODELS AND THE DATASET USED IN TRAINING THE MODELS

| | | CIFAR10 | IMGNET-A | IMGNET-B |
|---|---|---|---|---|
| Vanilla Model | Model-A | x | - | - |
| | Model-B | - | x | x |
| Transfer Model | Model-C | - | x | x |

TABLE III
THE ARCHITECTURE OF THE VANILLA MODEL FOR CIFAR10 DATASET (MODEL-A)

| Vanilla Model For CIFAR10 Dataset |
|---|
| Conv2D, Filter Size=3*3, No of Filters=32, Stride=1*1,Padding |
| Activation Layer (Relu) |
| Conv2D, Filter Size=3*3, No of Filters=32, Stride=1*1, Padding |
| Activation Layer (Relu) |
| Max Pooling, Pool Size = 2*2,Stride = 1*1, Padding |
| Dropout(0.25) |
| Conv2D, Filter Size = 3*3, No of Filters = 64,Stride = 1*1, Padding |
| Activation Layer (Relu) |
| Conv2D, Filter Size = 3*3, No of Filters = 64,Stride = 1*1, Padding |
| Activation Layer (Relu) |
| Max Pooling,Pool Size = 2*2,Stride = 1*1, Padding |
| Dropout(0.25) |
| Flatten |
| Dense(512) |
| Activation( Relu) |
| Dropout(0.5) |
| Dense(10) |
| Activation(Softmax) |

TABLE IV
THE ARCHITECTURE OF THE VANILLA MODEL FOR IMAGENET DATASET (MODEL-B)

| Vanilla Model For ImageNet Dataset |
|---|
| Conv2D, Filter Size = 3*3, No of Filters = 32, Stride = 1*1,No Padding |
| Activation Layer (Relu) |
| Max Pooling, Pool Size = 2*2, Stride = 1,1,No Padding |
| Conv2D, Filter Size = 3*3, No of Filters = 32, No Padding |
| Activation Layer (Relu) |
| Max Pooling, Pool Size = 2*2,Stride = 1*1,No Padding |
| Conv2D, Filter Size = 3*3, No of Filters = 32,Stride = 1*1, No Padding |
| Activation Layer (Relu) |
| Max Pooling,Pool Size = 2*2,Stride = 1*1,No Padding |
| Flatten |
| Dense(64) |
| Activation( Relu) |
| Dropout(0.5) |
| Dense(10) |
| Activation(Softmax) |

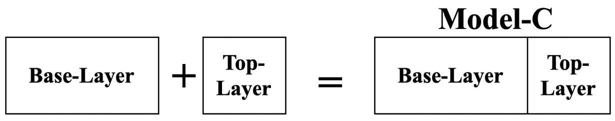


Fig. 6. The Transfer Learning Model Block (Model-C) .

all the images to obtain the latent variables needed to train the classifier. The mean square error (MSE), which also doubles as the cost function is used as the metrics of the autoencoder.

### C. Training Stage

*1) Classifier Design:* The convolutional neural network (CNN) model is used as the classifier in this work. CNNs are well suited for image processing applications and other grid-like data [15]. They are more computationally efficient than the dense deep neural network (DNN), thus reducing memory usage. Using the filters, CNNs find and extract meaningful features from the images and preserve spatial relations. Three different CNN classifiers, denoted Model-A, Model-B, and Model-C, as listed in Table II, are used in this work.

*Model-A and Model-B:* Model-A and Model-B are considered to be vanilla models because they are trained from scratch. Model-A and Model-B are specifically designed for the original input image and feature maps of the CIFAR10 dataset and ImageNet dataset, respectively. The detailed CNN architecture of Model-A and Model-B are shown in Tables IV and III, respectively. The models contain a mix of convolutional, max pooling, and fully connected layers. The ReLu and softmax activation functions are also used for the model design.

Furthermore, the models also contain some dropout layer in order to prevent over-fitting. The differences between Model-A and Model-B lie in the number of the various layers used and padding of the convolutional layers of Model-A.

The models are trained from scratch to minimize the difference between the labels (ground truth) and the predicted labels. This training is achieved by the use of the glorot-uniform method as the initializer, categorical cross-entropy as the loss function, and Adams optimization algorithm as the optimizer. Data augmentation is also used during the training process to mitigate overfitting due to the small quantity of the datasets. It should be stated that each classifier is trained with their respective original image and the feature maps (compressed images).

*Model-C:* Model-C is a transfer learning based model explicitly designed for the ImageNet dataset in this work. The CIFAR10 version of the result is not presented in this work as the compressed data gives a poor performance with the transfer learning models. This poor performance can be attributed to the small dimensions of the CIFAR10 dataset and large depth of the various transfer learning models used.

The block diagram of the model is shown in Fig. 6. Model-C can be divided into two parts: The base layer and the top layer. The base layer is a pre-trained layer of another standard deep learning model (without the fully connected layer) trained with data similar to the ImageNet data and achieved better performance. Using this pre-trained model, the excellent feature extracting property of the standard model is being leveraged to achieve better performance. Furthermore, it also complements data augmentation in training a decent model in situations where datasets are limited. VGG16, VGG19, InceptionV3, Inception-ResnetV2 and Resnet50 pre-trained models [25] are used as base models for Model-C.

The details of the top layer used for this work are shown in Table V. It should be noted that the first dense layer of the top layer in the fifth model (Model_5) is smaller than that of the other models. Model_5 suffers from overfitting if the number of neurons in the first layer is 256, the same number used in the other models. Hence, the size of the dense layer is lowered to reduce overfitting and achieve good performance.

A 2-stage training method is used for the transfer learning model to minimize the error between the ground truth labels and the predicted labels. This approach is different from the training

TABLE V
THE ARCHITECTURE OF THE TRANSFER LEARNING MODEL FOR IMAGENET DATASETS (MODEL-C)

| | **Model_1** | **Model_2** | **Model_3** | **Model_4** | **Model_5** |
|---|---|---|---|---|---|
| Base layer | VGG16 | VGG19 | InceptionV3 | InceptionResnetV2 | Resnet50 |
| Top layer | Dense(256) | Dense(256) | Dense(256) | Dense(256) | Dense(50) |
| | Activation(Relu) | Activation(Relu) | Activation(Relu) | Activation(Relu) | Activation(Relu) |
| | Dense(10) | Dense(10) | Dense(10) | Dense(10) | Dense(10) |
| | Activation(Softmax) | Activation(Softmax) | Activation(Softmax) | Activation(Softmax) | Activation(Softmax) |

approach used for Model-A and Model-B, which are trained from scratch. In the first stage, the base layer is fixed while the fully connected top layer is trained using the Adam optimizer after being initialized using the glorot-uniform method. This approach is taken to initialize the weight of the top layer close to the weight of the base layer. The complete model is then retrained, and all the weights are appropriately tuned using the stochastic gradient descent (SGD) with momentum optimizer. SGD with momentum is used because it is less aggressive than the Adam optimizer. The use of an aggressive optimizer in the second step might cause the weights (information) in the base layer to be significantly eroded or lost. The categorical cross-entropy is used as the cost function in the entire training process.

## D. Experiments

This work seeks to propose a new approach to design and implement deep learning models for distributed systems without compromising data privacy and security. It achieves this by extracting the most important/critical machine intelligible features but human unintelligible features from the dataset. These features are then transmitted across the communication network from the edge devices to the edge server, where they are aggregated and used to train a classifier. The experimental methods, performance metrics, and tools used in validating our proposed framework are explained in this section.

*1) Experimental Method:* A 2-stage methodology is used to validate our proposed framework, and this method is the same irrespective of the type of dataset or model used. In the first stage, the training set of the original input dataset (uncompressed images) is used to train the classifier. After that, the test set is used to obtain the needed performance metric to set the baseline performance.

In the second stage, the training set of the feature maps (compressed images of the dataset used in stage 1) is used to train the same classifier model. The feature map, which is smaller than the original image by a pre-determined factor, is obtained by passing the original dataset through the autoencoder's encoder. After that, the performance metric of the classifier is obtained using the test set of the feature maps and the performance compared to the baseline performance.

*2) Performance Metric:* The effectiveness of the framework is assessed using a simple classification task. The test accuracy of the model obtained after the training process is used as the primary performance metric although the F-score measurement of the models is also obtained to further validate the performance of the models. Furthermore, our proposed framework's effect on the training time, testing time, and the number of model parameters is also investigated. It should be noted that the primary performance metric for this section of the framework is application specific. For Natural language processing applications for example, this metric will change to either of Cosine Similarity, Jaccard Similarity, Perplexity or Word Error Rate.

*3) Software and Hardware:* The design, training, and testing of the deep learning models (Autoencoders and CNN Classifiers) are implemented using Keras deep learning framework on TensorFlow backend, running on an NVIDIA Tesla P100-PCIE-16 GB GPU.

## IV. RESULTS AND ANALYSIS

The results of the experimental work are presented in this section. The proposed framework's performance is compared with our baseline using the performance metrics stated in Section III-D2 above. The baseline performance is represented by compression ratio one (1), and it is synonymous with using the uncompressed image to test our various models. Furthermore, it should be noted that the vanilla model for the CIFAR10 and ImageNet datasets are different as stated in Section III-D.

Fig. 7 shows the testing accuracy of vanilla CNN Classifiers (Model-A and Model-B) when trained and tested with compressed and uncompressed CIFAR10 and ImageNet datasets. The testing accuracy for the compression ratio 1 (uncompressed images), representing our baseline, is highest across all the cases, as expected. This observation is because all the features in the raw images are used for the classification task. Furthermore, the testing accuracy for IMGNET-A is larger than the testing accuracy of IMGNET-B. The differences in performance can be attributed to the very close similarity in the images in IMGNET-B, as classifying such images is a much more difficult classification task than classifying images in IMGNET-A. The classifier requires more information than what is available to identify each of the class in IMGNET-B than IMGNET-A uniquely.

A general degradation in the testing accuracy is observed in Fig. 7 as the compression ratio is increased, although the rate of degradation varies across the models used for the three (3) datasets. The rate of degradation of the testing accuracy of the model trained with CIFAR10 dataset is the highest for all the compression ratios. The observed degradation is because the small dimension of the CIFAR10 images ($32 \times 32 \times 3$) implies that the number of features needed to perform a classification task is even smaller when compressed. This means the information contained in the image has been reduced, making it difficult for the model to have enough information to identify each class. Furthermore, the rate of degradation of the testing accuracy for the IMGNET-A dataset is very modest across all the compression ratios. However, similar performance is not observed in IMGNET-B, particularly for compression ratios 8 and

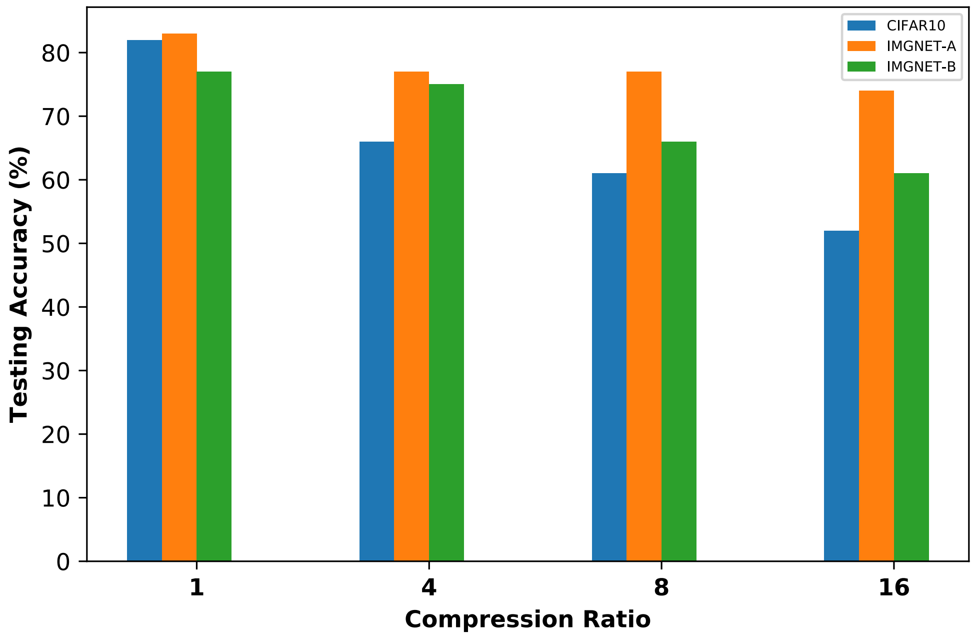


Fig. 7. Comparison of the testing accuracy of the vanilla models for the original dataset (compression ratio =1) and compressed dataset (latent variables) with compression ratio = 4, 8, and 16.

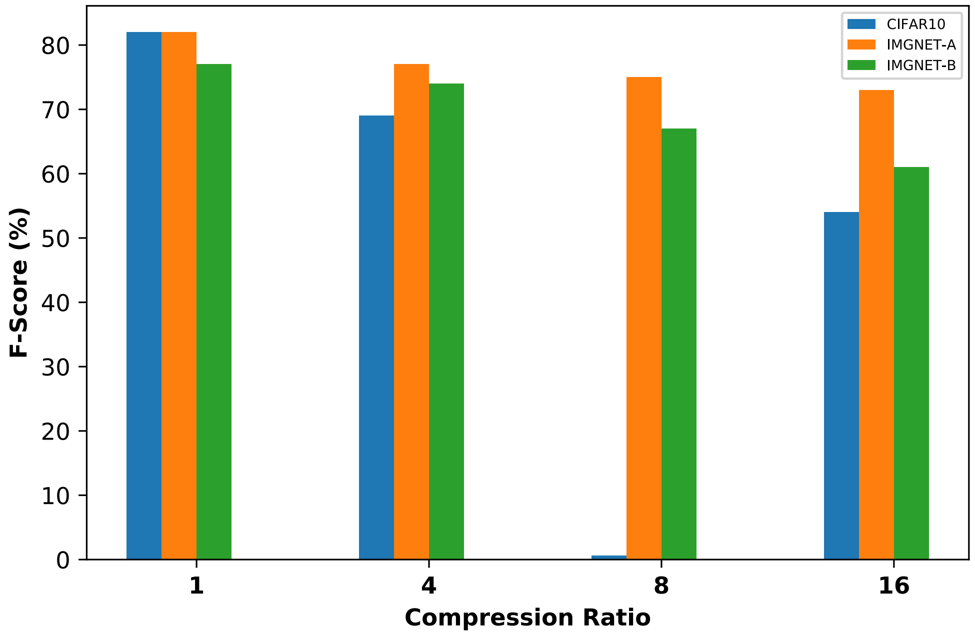


Fig. 8. Comparison of F-Score of the vanilla models for the original dataset (compression ratio =1) and compressed dataset (latent variables) with compression ratio = 4, 8, and 16.

16 despite having the same image dimension (224 × 224 × 3) as the IMGNET-A dataset. The larger degree of degradation observed in the model trained on IMGNET-B for compression ratios 8 and 16 is due to the complexity of the classification task. The degradation is because of the similarities in the images that make up the various classes in IMGNET-B, which indicates more features are needed to identify the image for each class uniquely. The considerable diversity in the IMGNET-A dataset classes makes the classification task less complex and requires fewer features to identify each class and differentiate between the classes uniquely. This reduced complexity explains why it suffers low degradation in testing accuracy even at a higher compression ratio despite fewer features being used for classification. In order to further validate the performance of the models, the F-score of the models is obtained and shown in Fig. 8. It can be observed that the F-score values of the models for various compression ratios are approximately the same as the corresponding model accuracy values. Furthermore, the F-score

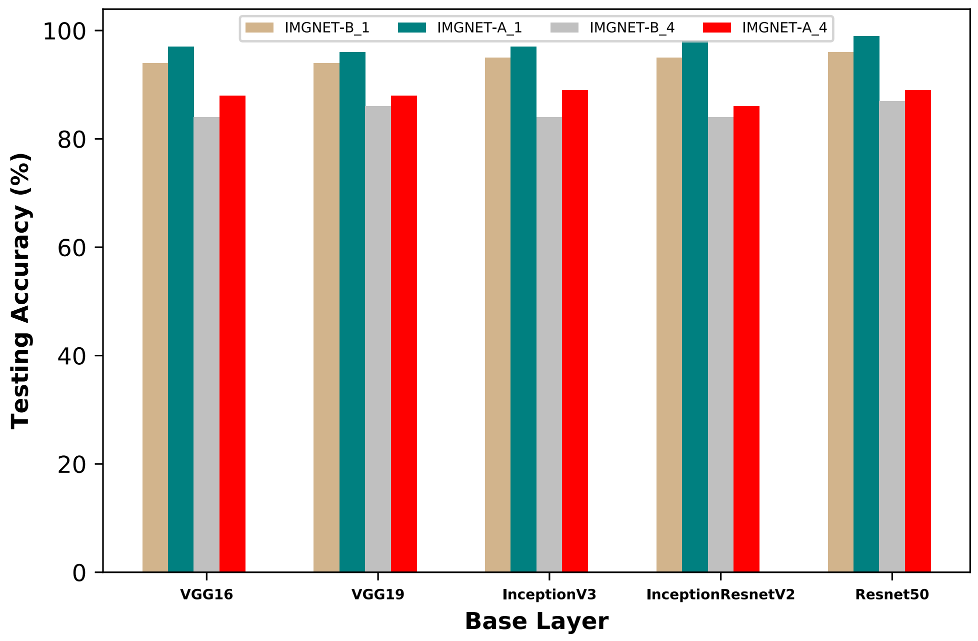


Fig. 9. Testing accuracy of the transfer learning based model (Model-C) using different base models for the ImageNet dataset with compression ratio = 4.

values show a similar trend as the testing accuracy when the compression ratio is increased. This is expected because the dataset used for this work is a balanced dataset.

The testing accuracy of the transfer learning based model (Model-C) for the ImageNet dataset compressed by a factor of 4, using different base models, is shown in Fig. 9. The transfer learning model is not designed and trained using the CIFAR10 datasets as its performance is poor with the compressed images. The poor performance can be attributed to the deep nature of the transfer learning based model. Due to its depth, the transfer learning model has an inadequate number of features available at the fully connected layer of the model (top layer) where classification takes place. Hence, there are inadequate features available for the classes in the dataset to be uniquely identified. The same reason also explains why the transfer learning model is only designed and tested with the ImageNet dataset with a compression ratio of four(4). At higher compression ratios (8 and 16), the performance is poor with the compressed images as there are fewer features at the fully connected layer of the model (top layer) for the model to uniquely identity each class in the IMGNET-A and IMGNET-B dataset.

From Fig. 7 and Fig. 9, it is observed that the testing accuracy of the transfer learning model across different base models for IMGNET-A and IMGNET-B datasets at compression ratio 1 (baseline) and 4 is higher than the corresponding performance of the vanilla model (Model-A and Model-B). This phenomenon can be attributed to the powerful feature extraction property of the various base layers used in the transfer learning model. The base layer is a neural network of various configurations or architectures trained on the complete ImageNet dataset for the classification task. The base layer has powerful feature extraction property because it has been trained on a classification task similar to the classification task at hand and achieves satisfactory testing accuracy.

As observed with the vanilla model in Fig. 7, the testing accuracy for compression ratio 1 (baseline) is better than the testing accuracy for compression ratio 4 for both IMGNET-A and IMGNET-B datasets with the transfer learning model as well (Fig. 9). Furthermore, the performance of the transfer learning model trained on the IMGNET-A dataset is better than that of the model trained on the IMGNET-B dataset for compression ratios one (1) and four (4). However, the rate of degradation in the testing accuracy for compression ratio 4 is higher than what is observed for the vanilla models (Fig. 7) for both IMGNET-A and IMGNET-B datasets. The degradation can also be attributed to the deep nature of the transfer learning models as the small amount of information/features available at the fully connected layer of the model (top layer) are not distinct enough to make an accurate classification. The number of parameters in a convolutional neural network is determined by many factors such as the filter size, the number of filters, the size of the input data, the number and type of hidden layers. Hence, a reduction in the number of trainable parameters can be achieved by reducing the size of the input data. The relationship between the normalized number of parameters in the vanilla model for the CIFAR10 and ImageNet datasets vs. the compression ratio is shown in Fig. 10. It can be observed that for the same compression ratio, the rate of reduction in the normalized number of parameters of the vanilla model for the ImageNet dataset is much bigger than that for the CIFAR10 dataset. This is because when the compression ratio increases, the already low-resolution images from CIFAR10 cannot be reduced much further by the model, and the number of parameters needed will remain almost constant for compression ratios 4, 8, and 16. On the contrary, the number of required parameters will keep decreasing in the case of the ImageNet dataset for compression ratio 4 and 8 because the images have much higher resolution, and as a result, the parameters will keep decreasing when the input image becomes smaller. However,

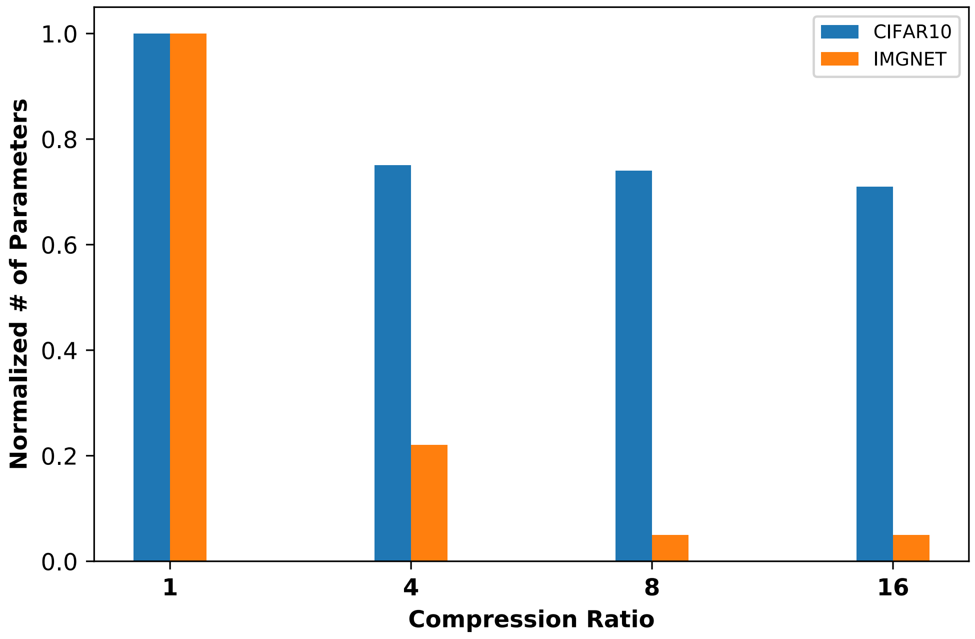


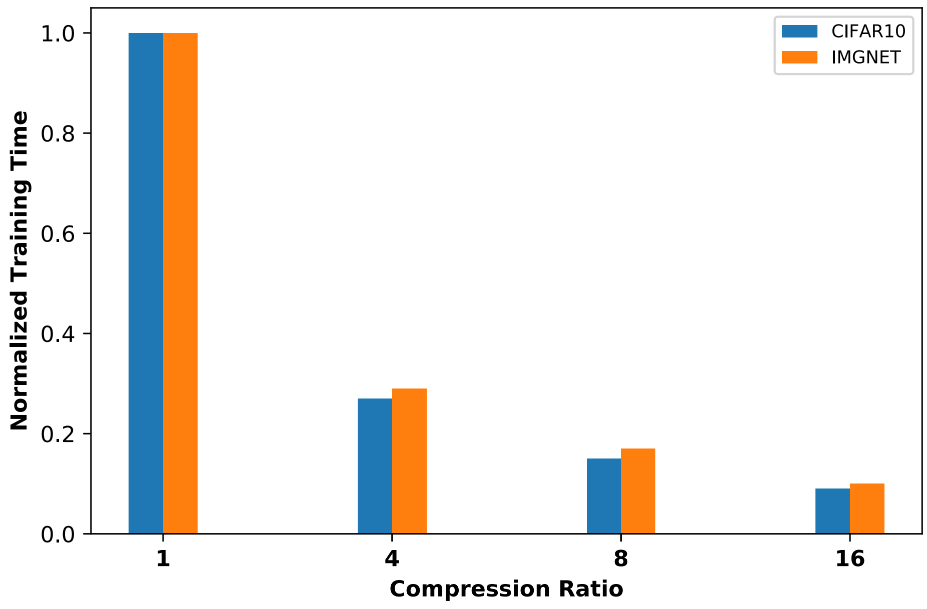
Fig. 10. Comparison of the normalized number of vanilla model parameters vs. data compression ratio.

when the compression ratio is 16 for ImageNet, the images are already small, they cannot be reduced much further, and thus will not affect the number of parameters.

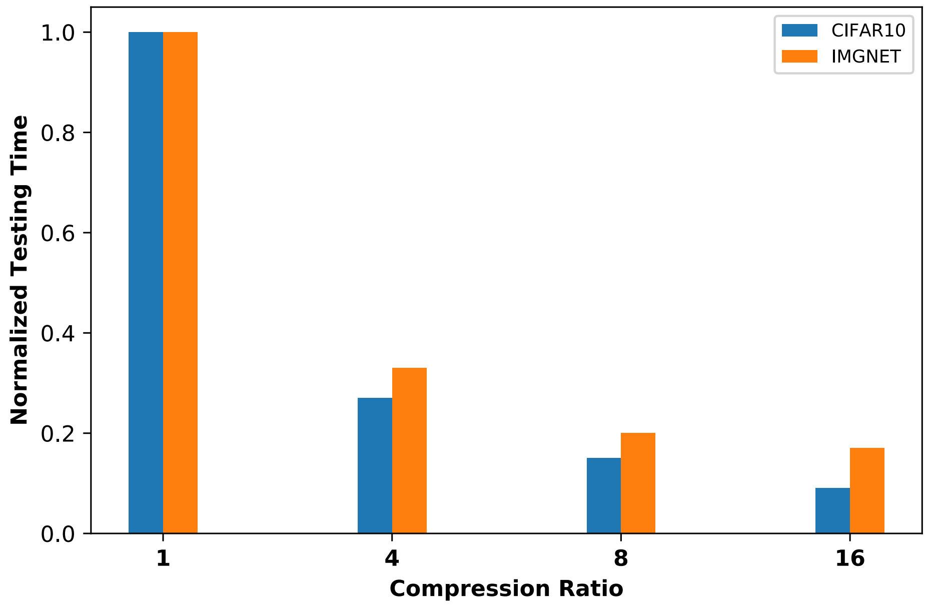
Fig. 11 shows the normalized amount of time required for testing and training of the vanilla models at various compression ratios for CIFAR10 and ImageNet datasets, respectively. A reduction in training and testing time is observed across the compression ratios and the models. The reduction can be attributed to the decreasing size of the input data, which directly affects the total number of trainable parameters. As the size of the input images decrease with the increase in compression factor, the total number of trainable parameters at the fully connected layer and the total number of trainable parameters also decrease. The total number of trainable parameters in a model indicate the degree of complexity of the resulting model and the amount of time required to train the model (training time) and test the model (testing time). Hence, the reduction in the training time and testing time as the compression factor increases.

Although Figs. 9-10 represents a desirable improvement in some of the properties of the resulting model, such as training time, testing time, and the number of parameters as the compression factor increases, the model accuracy, as shown in Figs. 7-8 reduces with increase in the compression factor. These Figures show the trade-off between the degree of privacy desired and the model accuracy. It also represents a trade-off between our proposed method and the compression factor of one, which is equivalent to server-only computation. Our proposed framework leverages on both the resources at the edge and at the cloud server. The edge computation part (autoencoder) helps extract the useful features needed for training at the cloud server. The cloud server provides the resource needed to train the deep learning model (classifier). Training only at the edge will lead to poor performance as there are no sufficient data and edge device cannot handle large deep model due to limited computing resource. Training only at the server will require all data available at the server, which might not be possible due to privacy concerns and limited bandwidth. Hence, there is a need to find an optimal trade-off point between the compression and model accuracy.

Although the testing accuracy is used as the primary metric for the framework in this work, this could change as the choice of this metric is application specific. The choice of testing accuracy as the primary metric in this paper is informed by the use of an image classification model as the second model. With object detection application, average precision or intersection over union metric can be used to judge the effectiveness of the model. In cases where a metric that is application dependent is desired, the use of the mean squared error (MSE), which is the cost function used in training the autoencoder can be considered. The MSE is the difference between the encoder input and the decoder output. This metric is the same irrespective of the type of application the framework is used for. The lower the MSE, the more the effectiveness of the encoder model in extracting the latent variable/features in the input. Furthermore, there is a strong negative correlation between the MSE and the testing accuracy. This is because the output of the pre-trained encoder is used in training the classification model in the second part of the model. Hence, the lower the MSE of the encoder, the better the accuracy of the resulting model trained with the output of the encoder. The plot of the MSE of the encoder used in generating the compressed input for Model-A and Model-B is shown in Fig. 12. It is noticed from the table that the MSE increases with an increase in the compression ratio for a particular dataset, which speaks to the inverse relationship between the MSE of the encoder and the testing accuracy of the classification model. It is also noticed that the MSE value of the encoders for ImageNet-A dataset is bigger than the MSE value of the encoders of ImageNet-B dataset. This same trend is noticed in Fig. 7 where



(a) Comparison of the normalized testing time



(b) Comparison of the normalized training time

Fig. 11. Comparison of the normalized testing time and training time of the vanilla models for various compression ratios for CIFAR10 and ImageNet datasets. (a) Comparison of the normalized testing time. (b) Comparison of the normalized training time.

the testing accuracy of the models trained with compressed images of ImageNet-A is bigger than those of ImageNet-B. This behavior is attributed to the degree of complexity of the images in ImageNet-B dataset due to their close similarity as compared to images in ImageNet-A.

## V. Discussions and Related Works

There is a growing trend of deploying powerful and advanced deep learning models to achieve state-of-the-art performance in processing IoT images and videos. The deployment of the deep learning model either only locally (on IoT edge device) or only at the server fit different scenarios [26]. On the one hand, IoT edge device only training/deployment is a good choice when the deep learning model is relatively small, and it does not suffer from data privacy and security concerns associated with sending data to the server. However, limited computational power, memory, and energy resource of IoT/mobile edge devices make it difficult to achieve good latency and energy consumption when training/inferencing on large models [26]. On the other hand, the server only deployment will provide help from edge servers to IoT/mobile devices via computation offloading to

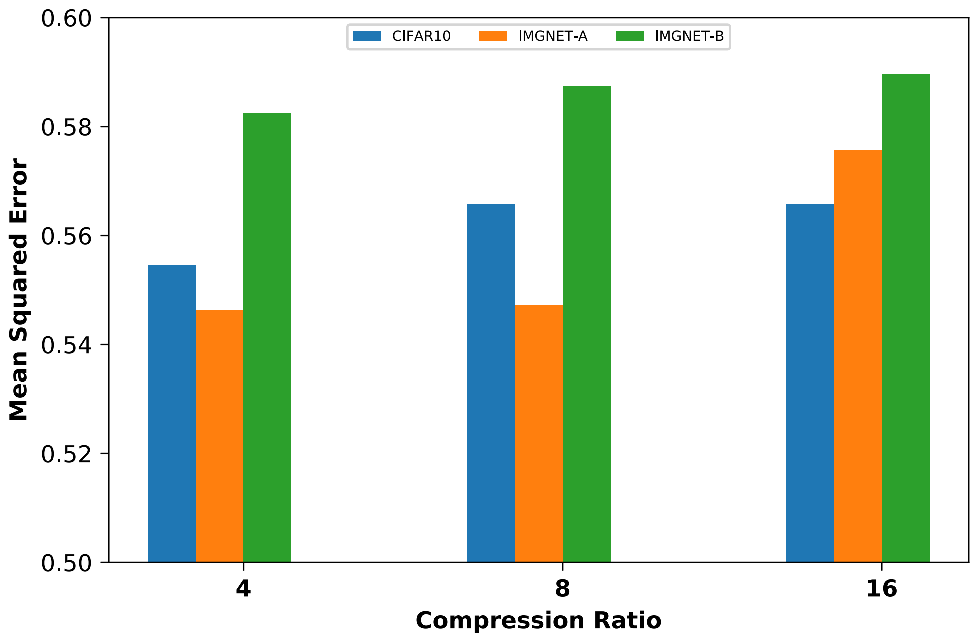


Fig. 12. Comparison of the mean squared error of the vanilla models for the original dataset (compression ratio =1) and the compressed dataset (latent variables) with compression ratio = 4, 8, and 16.

handle large models. Although server only deployment achieves scalability, low cost, and satisfactory quality of service (QoS), it suffers communication overhead for uploading the raw data and downloading the outputs, which consume much bandwidth and causes unpredictable latency due to the wireless channel [10], [26]. Also, privacy and security concerns are raised due to the transmission of raw data.

The desire to leverage on the merit of the server only and the IoT/mobile edge device only deployment of deep learning models has necessitated a new paradigm called collaborative intelligence, or collaborative training, or device-edge co-inference [11], [27]. In this new paradigm, the deep learning model is split between the edge device and the server as the computation required for earlier layers is done on the IoT/mobile edge device and the output of the layers called feature tensors are sent to the server for further processing [28]. Despite the advantages of collaborative intelligence in terms of less communication overhead and better data privacy, determining the optimal computation partition point in order to achieve reduced latency and edge device energy consumption is non-trivial because the choice of the best partition point depends on the system factors such as wireless channel state, computation capability of edge devices and edge servers and the deep learning model [10]. Many recent studies proposed various approaches to address this issue, such as [10], [27] Furthermore, it is possible to compress the intermediate features before sending them to the server instead of direct transfer [26], [28], [29].

Several methods are proposed in the literature to address the privacy and security concerns associated with data for training deep learning, such as homomorphic encryption [30], differential privacy [31], [32] and secure multiparty computation [33]. Furthermore, there are many authentication and key agreement schemes that have been proposed to ensure data privacy and security in IoT systems. An authentication framework that uses a digital certificate-based signature scheme that supports efficient signature operations with fast, modular arithmetic operations is proposed in [34]. The authors in [35] proposed ID-based cryptography (IBC) for authentication and the pseudonym-based mechanism for conditional privacy preservation and non-repudiation in urban vehicle communication. A similar authentication method for an edge-based smart grid environment which uses one-way hash functions, XOR computations, and an elliptic curve cryptosystem (ECC), is used in [36]. A framework that uses the cryptography based concept such as physically unclonable functions (PUF) and hash operations to achieve high levels of security at minimal computational resource cost, without requiring storage of security keys is proposed in [37]. Although a similar authentication scheme in [37] is proposed in [38], its uniqueness lies in the use of only one-way secure hash function and bitwise XOR operations for drone and users to authenticate each other. A multi-factor authenticated key establishment scheme based on the PUF, reverse fuzzy extractor, and cryptographic one-way hash function is proposed in [39] for secure smart grid communication.

In addition, some sanitation based methods have also been proposed to ensure data security and privacy. An ant colony system that uses a heuristic function based on pre-large concept and fitness function, with consideration for past selection and current situation, to reduce and monitor the side effects for a designated sanitation procedure is used in [40]. Although a similar ant colony optimization method is proposed in [41], it differs in that it uses multiple objective sanitation models and transaction deletion to hide and secure confidential and sensitive information. The method also uses pre-large conceptual model to reduce multiple scans of the database throughout the evaluation process to achieve lower computational cost. The work in [42] proposed a semantic privacy framework for the Internet of Medical Things, which improves the utility of the sanitized document by identifying negated assertions before the sanitation process and uses industry-standard metrics. Also, many particle swarm optimization (PSO) sanitation frameworks have been proposed. A hierarchical-cluster method, which uses a multi-objective particle swarm optimization framework to hide confidential information, balance the side effects while still discovering useful and meaningful information in the sanitized dataset, is proposed in [43]. The uniqueness of the PSO sanitation method in [44] lies in the use of the fitness function to minimize the side effects of sanitation by determining the maximum number of transactions to be deleted to efficiently hide sensitive itemsets and pre-large concept to speed up the evolution process. Similarly, PSO method with multiple thresholds and requires minimum support function threshold to hide sensitive information in a utility database is proposed in [45]. It should be noted that the proposed framework is not a substitute for sanitation or authentication methods. Authentication methods are used to establish trust between parties before data transmission to prevent unauthorized access or stealing of data. Data sanitation methods are used to hide confidential information by deleting it. This privacy preserving method is quite different from the aim of this study, where data might not be available to trusted parties due to privacy concerns. Authentication and data sanitation methods can still be used in conjunction with our proposed framework to provide an extra layer of privacy and security.

Despite the success of these methods, some issues remain, such as performance degradation, non-trivial overhead, or limited application [46]–[48]. The use of collaborative deep learning method, such as federated learning and SplitNN, in distributed learning, has been introduced in recent years to solve the problem of data privacy. Federated learning is a type of machine learning where the goal is to train a high-quality centralized model while the data remains distributed over a large number of clients [3]. It involves sharing model parameters and model gradients through a parameter server without sharing their local data. Federated learning is based on an iterative model averaging, and it is robust to unbalanced data and non-i.i.d. data distribution. Federated learning has been applied to mobile keyboard prediction, vocabulary word learning, and google keyboard query suggestions improvement [49]–[51]. Federated learning may be viewed as an extension of the idea discussed in [52], [53] that stochastic gradient descent can be implemented in parallel and asynchronously. Federated learning may suffer from non-trivial communication cost. To deal with the high communication cost, an efficient multi-objective evolutionary algorithm, based on a scalable network connectivity encoding method, is proposed in [5]. The use of structured and sketched updates are introduced in [4] to help reduce the uplink communication bottleneck.

Federated learning may also suffer from security/privacy issues due to the need to communicate the model parameters to the central server. One recent study showed potential security/privacy issues due to the possibility of reconstructing original data from the shared gradient [54]. Secure aggregation, a type of secure multi-party computation algorithm for federated learning, is introduced in [55]. Secure aggregation helps guarantee the privacy of data used in generating gradients shared by each model and improving communication efficiency. Furthermore, it is observed that federated learning performs poorly when the data distributed across the training center is strictly non-i.i.d. of a single class. This statistical challenge is resolved by creating and using a small subset of globally shared data between all the edge devices [56] or adopting a multi-task learning approach [57].

SplitNN can be considered to be a form of collaborative intelligence in a distributed learning environment [7], [58]. The edge device trains the first sub-network up to the cut layer and sends the intermediate features to the server, and the server processes the second sub-network using the received features, a process known as forward propagation. In turn, the server generates the gradient for the final layer, back-propagates the error up to the cut-layer, and sends the relevant gradients to the edge device. The edge device then uses the received gradient to generate the required gradient needed to update the weight [59]. In cases where there is more than one client, the training of the model is done sequentially, which is different from federated learning where the training is done in parallel [59].

The first work on SplitNN is done in [7] where its performance is compared with large SGD and federated learning. It established that SplitNN achieves a significantly lower computational burden on clients and lower communication cost during training than other distributed learning methods. Furthermore, it also showed that SplitNN achieves faster convergence than federated learning when there are many clients. The work on SplitNN in [7] is extended in [60] to use all of the partial clients-networks on each iteration sequentially. This method is suited for vertically partitioned data. It is achieved with each client computes a fixed portion of the computation graph and passes it to the server. The server computes the rest and performs back-propagation, and returns back the Jacobians to the client. Then the client can perform their respective back-propagation. The use of SplitNN to demonstrate the importance of collaborative training of deep learning model using health data is demonstrated in [61], [62]. In [61] several novel configurations of SplitNN are introduced. It also established that SplitNN achieves higher accuracies than that of other distributed learning methods on classification tasks and drastically lowers computational requirements on the client's side. Furthermore, SplitNN requires lower communication bandwidth than federated learning when there are a more significant number of clients.

A comparison between collaborative and non-collaborative training modes is carried out, and the impact of the number of clients on the performance of both modes is investigated in [62]. The privacy property of SplitNN is enhanced in [63] by minimizing the distance correlation between the intermediate features and the input data to reduce leakage. The empirically evaluation and comparison of both federated learning and SplitNN using imbalanced data and non-independent and identically distributed (non-IID) data using real-world IoT settings for performance and overhead (training time, communication overhead, power consumption, and memory usage) is shown in [64]. To leverage on the advantages of federated learning and SplitNN, SplitFed (SFL), a combination of both approaches to eliminate their inherent drawbacks, is introduced in [59].

Compared to collaborative intelligence, where the size of the intermediate feature tensor at the cut layer or optimal layer might be bigger than the original input features, the size of the encoder's intermediate feature tensor in the proposed framework is always smaller than its original input. The encoder compresses the original data to latent vectors, and the compression ratio can be controlled to provide additional flexibility. Uploading of compressed intermediate features to the server leads to lower transmission latency and energy consumption [10], [11]. Although it is possible to compress the intermediate features before sending them to the server in collaborative intelligence, an additional compression step must occur. On the contrary, the proposed autoencoder will extract salient features as latent vectors and perform controlled compression at the same time without the need for additional steps for compression.

In SplitNN, the training of the model is done sequentially. As a result, the training may be very time consuming when the number of edge devices is significant. On the contrary, the autoencoders at the edge devices can be trained in parallel in the proposed framework. Constant communication is also required to exchange intermediate features and gradients in SplitNN, which incurs much overhead and high communication cost. In the proposed framework, the edge device will send the latent vectors only once for the server to train the CNN classifier.

Autoencoder has been applied to address data privacy concerns in several recent works [2], [65], [66]. In [65], a convolutional autoencoder that perturbs an input face image to impart privacy to a subject is proposed. It is shown the method can protect gender privacy of face images. A proof-of-concept study has been performed in [2] to use an autoencoder for preserving video privacy, especially when non-healthcare professionals are involved. A modified sparse denoising autoencoder has been applied in [66] to reduce the sparsity and denoise the data. A 3-class classification is performed on the reconstructed data obtained from the autoencoder, and it is shown that the classifier can classify the original black class data as the transformed gray class data.

Although autoencoder has been used to address data privacy concerns, this work is the first to use autoencoder for addressing privacy concerns, communication cost, and deep learning efficiency associated with mobile edge computing systems with a large number of edge devices. The enhanced privacy is achieved using the autoencoder to extract human unintelligible but machine intelligible features from the data. The features or latent vectors are then used to train the classifier. Furthermore, the proposed approach comes with the added advantage of reducing the dimensionality of data needed to be transmitted, reducing the communication cost and the number of model parameters, training time, and inference time. This approach does not suffer from leaking gradient problem associated with federated learning [54] or increase in the size of the intermediate features early on in the models as sometimes observed in SplitNN [10].

## VI. Conclusion

Edge intelligent computing is an important emerging research topic with the deployment of 5 G and IoT in recent years. At the same time, privacy preservation of users is indispensable. The proposed framework encourages novel design and implementation of efficient privacy-preserving edge intelligent computing. The proposed framework provides 1) flexibility of training autoencoder at each edge device individually, thus protect the data privacy of end-users because raw data is not transmitted at any time; 2) after the training of autoencoder is complete, raw data is "compressed" and "encrypted" by the encoder before transmitting to the edge server, and this will reduce the communications cost, and further protect the data privacy and security; 3) the autoencoder will provide features to the classifier at the server, thus allow the classifier to be trained on the features with less and controlled dimensions; 4) the decoupling of the training of the autoencoder at the edge devices and the training of the classifier at the edge server relaxes the frequent communications requirement between edge devices and edge server. Experiments have been carried out using CIFAR10 and ImageNet datasets. A detailed analysis of the tradeoff between classifier accuracy, the dimensionality of data, compression ratio, and various choices of classifiers has been given to provide benchmarks and insights on the proposed scheme. To the best of our knowledge, this is the first attempt to design a framework to address the image classification problem in an edge computing scenario, where an autoencoder is designed to compress the raw images and extract salient features at the same time. In the paper, the proposed framework has been compared to the uncompressed approach (compression ratio $= 1$), which can be considered the baseline model. In addition, all the transfer learning models are indeed state-of-the-art. Combining them with the proposed framework will result in a highly efficient and privacy-preserving edge intelligent computing solution. For future work, comparison with federated learning and SplitNN in terms of classifier performance vs. the communications cost and model complexity will be carried out for image classification tasks. The comparison will help quantify the advantages and disadvantages of the proposed approach. Furthermore, the use of other types of autoencoder to extract latent variables and the use of knowledge distillation to help mitigate the reduction in the model accuracy will be explored.

## Acknowledgment

The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. National Science Foundation or the Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)) or the U.S. Government.

## REFERENCES

[1] R. Ramesh, "Predictive analytics for banking user data using AWS machine learning cloud service," in *Proc. 2nd Int. Conf. Comput. Commun. Technol.*, 2017, pp. 210–215.

[2] M. D'Souza *et al.*, "Autoencoder - A new method for keeping data privacy when analyzing videos of patients with motor dysfunction (P4.001)," *Neurology*, vol. 90, no. 15 Supplement, 2018. [Online]. Available: https://n.neurology.org/content/90/15_Supplement/P4.001

[3] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," 2016. [Online]. Available: http://arxiv.org/abs/1602.05629

[4] J. Konecny, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *Proc. NIPS Workshop Private Multi-Party Mach. Learn.*, 2016. [Online]. Available: https://arxiv.org/abs/1610.05492

[5] H. Zhu and Y. Jin, "Multi-objective evolutionary federated learning," 2018. [Online]. Available: http://arxiv.org/abs/1812.07478

[6] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3298981

[7] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, 2018.

[8] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," 2017. [Online]. Available: http://arxiv.org/abs/1712.07557

[9] K. Bonawitz *et al.*, "Practical secure aggregation for federated learning on user-held data," in *Proc. NIPS Workshop Private Multi-Party Mach. Learn.*, 2016.

[10] W. Shi, Y. Hou, S. Zhou, Z. Niu, Y. Zhang, and L. Geng, "Improving device-edge cooperative inference of deep learning via 2-step pruning," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2019, pp. 1–6.

[11] J. Shao and J. Zhang, "Bottlenet : An end-to-end approach for feature compression in device-edge co-inference systems," in *Proc. IEEE Int. Conf. Commun. Workshops*, 2020, pp. 1–6.

[12] " Google Edge Tpu," Accessed: Aug. 12, 2020. [Online]. Available: https://coral.ai/docs/edgetpu/faq/

[13] "Jetson Nano Developer Kit," Accessed: Oct. 14, 2019. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[14] M. Maggipinto, C. Masiero, A. Beghi, and G. A. Susto, "A convolutional autoencoder approach for feature extraction in virtual metrology," in *Proc. 28th Int. Conf. Flexible Automat. Intell. Manuf.*, Jun. 11–14, 2018, Columbus, OH, USA, pp. 126–133. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2351978918311399

[15] I. J. Goodfellow, Y. Bengio, and A. C. Courville, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[16] D. Adesina, J. Bassey, and L. Qian, "Robust deep radio frequency spectrum learning for future wireless communications systems," *IEEE Access*, vol. 8, pp. 148528–148540, 2020.

[17] Z. Yu *et al.*, "A deep convolutional neural network-based framework for automatic fetal facial standard plane recognition," *IEEE J. Biomed. Health Informat.*, vol. 22, no. 3, pp. 874–885, May 2018.

[18] D. Sahoo, Q. Pham, J. Lu, and S. C. H. Hoi, "Online deep learning: learning deep neural networks on the fly," 2017. [Online]. Available: http://arxiv.org/abs/1711.03705

[19] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. A. Gama, "Machine learning for streaming data: state of the art, challenges, and opportunities," *ACM SIGKDD Explorations Newslett.*, vol. 21, no. 2, pp. 6–22, Nov. 2019. [Online]. Available: https://doi.org/10.1145/3373464.3373470

[20] I. Kontopoulos, A. Makris, and K. Tserpes, "A deep learning streaming methodology for trajectory classification," *ISPRS Int. J. Geo- Inf.*, vol. 10, no. 4, pp. 250, 2021. [Online]. Available: https://www.mdpi.com/2220-9964/10/4/250

[21] A. Dosovitskiy and T. Brox, "Inverting visual representations with convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 4829–4837.

[22] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 5188–5196.

[23] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, 2012.

[24] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[25] F. Chollet, *Deep Learning With Python*, *1st ed. USA*: Manning Publications Co., 2017.

[26] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 565–576, Feb. 2021.

[27] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGARCH Comput. Architecture News*, vol. 45, no. 1, pp. 615–629, Apr. 2017. [Online]. Available: https://doi.org/10.1145/3093337.3037698

[28] A. E. Eshratifar, A. Esmaili, and M. Pedram, "Bottlenet: A deep learning architecture for intelligent mobile cloud computing services," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des.*, 2019, pp. 1–6.

[29] H. Choi and I. Bajic, "Near-lossless deep feature compression for collaborative intelligence," in *Proc. IEEE 20th Int. Workshop Multimedia Signal Process.*, 2018, pp. 1–6.

[30] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. E. Lauter, and M. Naehrig, "Crypto-nets: Neural networks over encrypted data," 2014, *arXiv:1412.6181*.

[31] C. Dwork, "Differential privacy: A survey of results," in *Theory and Applications Models Computation*, M. Agrawal, D. Du, Z. Duan, and A. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–19.

[32] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.

[33] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," *IACR Cryptol. ePrint Arch.*, vol. 2017, pp. 619–631, 2017.

[34] Y. Tian, J. Yuan, and H. Song, "Efficient privacy-preserving authentication framework for edge-assisted internet of drones," *J. Inf. Secur. Appl.*, vol. 48, 2019, Art. no. 102354. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212618307038

[35] J. Li, H. Lu, and M. Guizani, "ACPN: A novel authentication framework with conditional privacy-preservation and non-repudiation for VANETs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 938–948, Apr. 2015.

[36] C.-M. Chen, L. Chen, Y. Huang, S. Kumar, and J. M.-T. Wu, "Lightweight authentication protocol in edge-based smart grid environment," *EURASIP J. Wireless Commun. Netw.*, vol. 68, 2021.

[37] P. Gope and B. Sikdar, "An efficient privacy-preserving authenticated key agreement scheme for edge-assisted internet of drones," *IEEE Trans. Veh. Technol.*, vol. 69, no. 11, pp. 13621–13630, Nov. 2020.

[38] Y. Zhang, D. He, L. Li, and B. Chen, "A lightweight authentication and key agreement scheme for internet of drones," *Comput. Commun.*, vol. 154, pp. 455–464, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366419319358

[39] P. Gope, "Pmake: Privacy-aware multi-factor authenticated key establishment scheme for advance metering infrastructure in smart grid," *Comput. Commun.*, vol. 152, pp. 338–344, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366419313210

[40] J. M.-T. Wu, J. Zhan, and J. C.-W. Lin, "Ant colony system sanitization approach to hiding sensitive itemsets," *IEEE Access*, vol. 5, pp. 10024–10039, 2017.

[41] J. C.-W. Lin, G. Srivastava, Y. Zhang, Y. Djenouri, and M. Aloqaily, "Privacy-preserving multiobjective sanitization model in 6G IoT environments," *IEEE Internet Things J.*, vol. 8, no. 7, pp. 5340–5349, Apr. 2021.

[42] "N-sanitization: A semantic privacy-preserving framework for unstructured medical datasets," *Comput. Commun.*, vol. 161, pp. 160–171, 2020.

[43] J. C.-W. Lin *et al.*, "A sanitization approach to secure shared data in an IoT environment," *IEEE Access*, vol. 7, pp. 25359–25368, 2019.

[44] J. C.-W. Lin, Q. Liu, P. Fournier-Viger, T.-P. Hong, M. Voznak, and J. Zhan, "A sanitization approach for hiding sensitive itemsets based on particle swarm optimization," *Eng. Appl. Artif. Intell.*, vol. 53, pp. 1–18, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197616300653

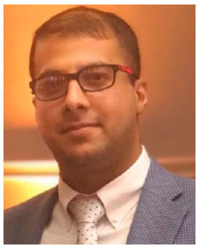
[45] J. M.-T. Wu, G. Srivastava, U. Yun, S. Tayeb, and J. C.-W. Lin, "An evolutionary computation-based privacy-preserving data mining model under a multithreshold constraint," *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 3, 2021, Art. no. e4209. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4209

[46] H. Bae, J. Jang, D. Jung, H. Jang, H. Ha, and S. Yoon, "Security and privacy issues in deep learning," 2018, *arXiv:1807.11655*.

[47] J. Zhao, Y. Chen, and W. Zhang, "Differential privacy preservation in deep learning: Challenges, opportunities and solutions," *IEEE Access*, vol. 7, pp. 48901–48911, 2019.

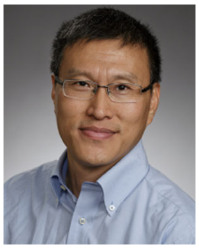
[48] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, 2015, pp. 1310–1321. [Online]. Available: https://doi.org/10.1145/2810103.2813687
[49] T. Yang *et al.*, "Applied federated learning: Improving google keyboard query suggestions," 2018. [Online]. Available: http://arxiv.org/abs/1812.02903
[50] A. Hard *et al.*, "Federated learning for mobile keyboard prediction," 2018. [Online]. Available: http://arxiv.org/abs/1811.03604
[51] M. Chen, R. Mathews, T. Ouyang, and F. Beaufays, "Federated learning of out-of-vocabulary words," 2019. [Online]. Available: http://arxiv.org/abs/1903.10635
[52] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.* 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1223–1231. [Online]. Available: http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf
[53] J. Chen, R. Monga, S. Bengio, and R. Józefowicz, "Revisiting distributed synchronous SGD," 2016. [Online]. Available: http://arxiv.org/abs/1604.00981
[54] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," 2019. [Online]. Available: http://arxiv.org/abs/1906.08935
[55] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.
[56] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," 2018. [Online]. Available: http://arxiv.org/abs/1806.00582
[57] V. Smith, C. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," 2017. [Online]. Available: http://arxiv.org/abs/1705.10467
[58] P. Vepakomma, T. Swedish, R. Raskar, O. Gupta, and A. Dubey, "No peek: A survey of private distributed deep learning," 2018, *arXiv:1812.03288*.
[59] C. Thapa, M. Chamikara, and S. Camtepe, "Splitfed: When federated learning meets split learning," 2020, *arXiv:2004.12088*.
[60] I. Ceballos *et al.*, "Splitnn-driven vertical partitioning," 2020, *arXiv:2008.04137*.
[61] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," 2018, *arXiv:1812.00564*.
[62] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar, "Split learning for collaborative deep learning in healthcare," 2019, *arXiv:1912.12115*.
[63] P. Vepakomma, O. Gupta, A. Dubey, and R. Raskar, "Reducing leakage in distributed deep learning for sensitive health data," 2019, *arXiv:1812.00564*.
[64] Y. Gao *et al.*, "End-to-end evaluation of federated learning and split learning for Internet of Things," in *Proc. Int. Symp. Reliable Distrib. Syst.*, 2020, pp. 91–100.
[65] V. Mirjalili, S. Raschka, A. Namboodiri, and A. Ross, "Semi-adversarial networks: Convolutional autoencoders for imparting privacy to face images," in *Proc. Int. Conf. Biometrics*, 2018, pp. 82–89. [Online]. Available: http://dx.doi.org/10.1109/ICB2018.2018.00023
[66] R. M. Alguliyev, R. M. Aliguliyev, and F. J. Abdullayeva, "Privacy-preserving deep learning algorithm for big personal data analysis," *J. Ind. Inf. Integration*, vol. 15, pp. 1–14, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2452414X18301456

**Omobayode Fagbohungbe** (Student Member, IEEE) received the B.S. degree in electronic and electrical engineering from Obafemi Awolowo University, Ile-Ife, Nigeria, and the M.S. degree in control engineering from the University of Manchester, Manchester, U.K. He is currently working toward the Ph.D. degree from the U.S. DOD Center of Excellence in Research and Education for Big Military Data Intelligence (CREDIT Center), Department of Electrical and Computer Engineering, Prairie View A&M University, Prairie View, TX, USA. His research interests include Big Data, data science, robust deep learning models, and artificial intelligence.



**Sheikh Rufsan Reza** (Student Member, IEEE) received the B.S. degree in electrical and electronics engineering from North South University, Dhaka, Bangladesh, in 2013 and the M.S. degree in electrical engineering from the University of Massachusetts Lowell, MA, USA, in 2016. He is currently working toward the Ph.D. degree in electrical engineering with Prairie View A&M University, Prairie View, TX, USA. His research interests include implementing lighter and robust machine learning models compatible for resource constraint devices for computer vision tasks.

**Xishuang Dong** (Member, IEEE) received the B.E. degree in computer science and technology from the Harbin University of Science and Technology, Harbin, China, the M.E. degree in computer software and theory from Harbin Engineering University, Harbin, and the Ph.D. degree in computer application technology from the Harbin Institute of Technology, in 2005, 2008, and 2013, respectively. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Prairie View A&M University, Prairie View, TX, USA. His research interests include deep learning, computational systems biology, and natural language processing.



**Lijun Qian** (Senior Member, IEEE) received the B.S. degree from Tsinghua University, Beijing, China, the M.S. degree from the Technion - Israel Institute of Technology, Haifa, Israel, and the Ph.D. degree from Rutgers University, USA. He is currently a Regents Professor and holds the AT&T Endowment with the Department of Electrical and Computer Engineering, Prairie View A&M University (PVAMU), Prairie View, TX, USA, and a Member of Texas A&M University System, Prairie View, TX, USA. He is also the Director of the Center of Excellence in Research and Education for Big Military Data Intelligence (CREDIT Center). Before joining PVAMU, he was a Member of technical staff of Bell-Labs Research, Murray Hill, NJ, USA. He was a Visiting Professor with Aalto University, Finland. His research interests include big data processing, artificial intelligence, wireless communications and mobile networks, network security and intrusion detection, and computational and systems biology.