

# Adaptive Vertical Federated Learning via Feature Map Transferring in Mobile Edge Computing

Yuanzhang Li<sup>1</sup>, Tianchi Sha<sup>1</sup>, Thar Baker<sup>2\*</sup>, Xiao Yu<sup>3\*</sup>, Zhiwei Shi<sup>4</sup> and Sikang Hu<sup>1</sup>

<sup>1</sup>School of Computer Science and Technology, Beijing Institute of Technology, Beijing, 100081, Beijing, China.

<sup>2\*</sup>Department of Computer Science, University of Sharjah, UAE.

<sup>3\*</sup>Department of Computer Science and Technology, Shandong University of Technology, Zibo, 255022, China.

<sup>4</sup>China Information Technology Security Evaluation Center, Zibo, 100085, Beijing, China.

\*Corresponding author(s). E-mail(s): [tshamsa@sharjah.ac.ae](mailto:tshamsa@sharjah.ac.ae);  
[yuxiao8907118@163.com](mailto:yuxiao8907118@163.com);

Contributing authors: [popular@bit.edu.cn](mailto:popular@bit.edu.cn); [412452812@qq.com](mailto:412452812@qq.com);  
[shizhiweicp@126.com](mailto:shizhiweicp@126.com); [skhu@bit.edu.cn](mailto:skhu@bit.edu.cn);

## Abstract

To bring more intelligence to edge systems, Federated Learning (FL) is proposed to provide a privacy-preserving mechanism to train a globally shared model by utilizing a massive amount of user-generated data on devices. FL enables multiple clients collaboratively train a machine learning model while keeping the raw training data local. When the dataset is horizontally partitioned, existing FL algorithms can aggregate CNN models received from decentralized clients. But, it cannot be applied to the scenario where the dataset is vertically partitioned. This manuscript showcases the task of image classification in the vertical FL settings in which participants hold incomplete image pieces of all samples, individually. To this end, the paper discusses AdptVFedConv to tackle this issue and achieves the CNN models' aim for training without revealing raw data. Unlike conventional FL algorithms for sharing model parameters in every communication iteration, AdptVFedConv enables hidden feature representations. Each client

fine-tunes a local feature extractor and transmits the extracted feature representations to the backend machine. A classifier model is trained with concatenated feature representations as input and ground truth labels as output at the server-side. Furthermore, we put forward the model transfer method and replication padding tricks to improve final performance. Extensive experiments demonstrate that the accuracy of AdptVFedConv is close to the centralized model.

**Keywords:** Federated learning, Transfer learning, Convolutional neural network, Machine learning

## 1 Introduction

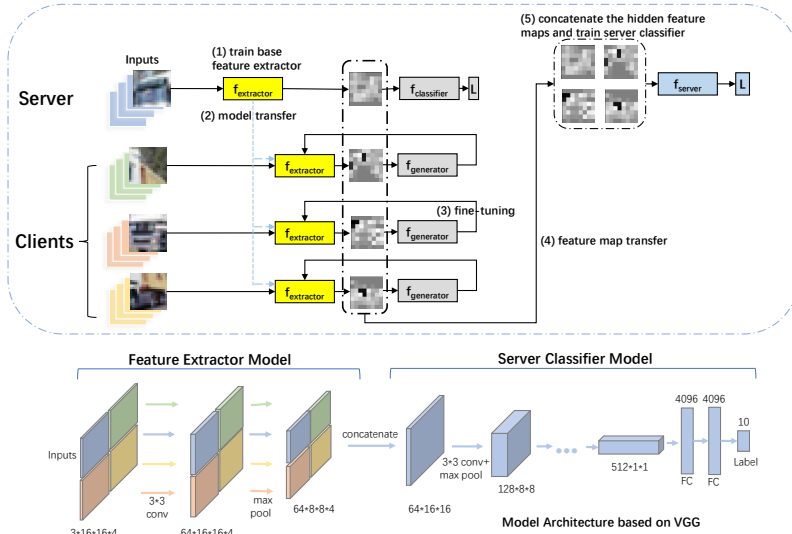
Along with the rapid development of mobile edge computing technology, more intelligence is brought to edge systems which can significantly bridge the capacity of the cloud and the requirement of devices and thus can boost the response performance of devices and improve the quality of mobile services. While the successful applications of intelligence in mobile edge computing, more concerns have been raised about privacy problems. Federated Learning (FL) has been coined to help decentralized data sources to jointly train a machine learning model while keeping privacy data local. In the scenario where a single organization or user cannot collect adequate data, FL can help improving accuracy. According to different application scenarios, FL can be categorized as horizontal FL, vertical FL, and federated transfer learning [1].

Horizontal FL has been used in scenarios where datasets share the same (or, similar) feature space from various space sample. For example, two hospitals in one city serve different patients and record similar body information. They can use horizontal FL approaches to collaboratively train a medical image classification model. Whereas in vertical FL, datasets of different clients share the same sample space from different feature space. For example, an e-commerce company and a bank in one city may serve the same users, but the feature space of their collected datasets is different. The e-commerce company records the users' online shopping behaviors while the bank has the users' deposit and loan information. Training a model through a vertical FL method can assist in better evaluation of users' credit ratings.

Numerous techniques have been discussed in the literature in the context of privacy-preserving vertical FL such as [2, 4–9]. Vertical FL algorithms are based mainly on security mechanisms because each participant cannot achieve the local training objective independently. Common security mechanisms include homomorphic encryption and secure multi-party computation.

The core issue of FL is how to improve the performance of machine learning models through communication. Existing algorithms average uploaded model parameters to get a global model. However, such algorithms suffer from these disadvantages: (i) Modern ML models, especially artificial neural networks, may memorize arbitrary information. Sharing model parameters may also take the risk of revealing some information about the raw training data. Recent works have demonstrated that FL may not always provide sufficient privacy guarantees, as sharing

model parameters can nonetheless reveal sensitive information [10–15]. (ii) Each client sample from a unique data distribution making the datasets heterogeneous, i.e., non-independent identically distributed (non-iid). Work [16] showed that, with non-iid datasets, the performance of conventional FL algorithms degrade significantly. (iii) With neural networks becoming deeper, the communication cost of exchanging model parameters increases.



**Fig. 1:** The Pipeline of AdptVFedConv. Each sample in the dataset is divided into a  $2 \times 2$  grid. These 4 tiles are held by 4 different participants. The server owns the ground truth label. First, the server trains a base model and shares it with clients. Each client fine-tunes the local feature extractor with local data and uploads their hidden feature representations. The server will collect and concatenate them to train a server classifier.

To overcome these disadvantages, we seek inspiration from human behavior. Human beings can learn new knowledge through communication in a different way from FL. When communicating with others, we neither share raw information we received nor aggregate the brain’s “parameters”. In contrast, we transform received information into language that can be understood by others. In artificial neural networks, the model is similar to a human’s brain and we can regard hidden layer outputs as language. Thus, in the proposed framework, participants share the hidden feature maps instead of model parameters. Hidden feature maps carry feature information of training data so that other models can also take it as inputs. But it is hard to restore the training data without knowing model parameters.

Let hidden feature maps take the function of language, one challenge is to estimate which layer’s output is optimal. Taking image recognition as an example, when we see a painting and try to describe objects in it, the content expressed by different people is always similar and can be understood by others. But when we discuss

the idea the painting wants to convey, different people often come to different conclusions. Human beings always get similar conclusions on common sense problems which means the model used in this step is similar for different people. Inspired by these, we divide a neural network into a shallow model and a deep model. The shallow model, also called the local feature extractor, can extract feature maps from input data. The deep model, also called the server classifier, can take feature maps collected from all participants as input to predict the correct label.

Figure 1 illustrates how AdptVFedConv works. The dataset is vertically partitioned into several grids and each client owns one of them. We succeed the case study presented by Yang [1] when a single positive participant owns labels and can't reveal them to anyone. The proposed model architecture is based on VGG. The later is broken up into the server part and the client part.

To this end, the key contributions of this manuscript are the following:

1. Present a novel adaptive vertical federated learning algorithm called AdptVFedConv. In the algorithm, the feature extractor can be used to train a global feature classifier at the server-side or train a personalized feature classifier to adapt to local data distribution. When clients get the base feature extractor from the backend, they train a local autoencoder and adaptively fine-tune the feature extractor with local data. By reducing the loss between input image and generated image, the feature extractor adapts to local data distribution.
2. Compare AdptVFedConv with the centralized model and analyze the causes of accuracy loss. We then propose several optimization methods. We rely on the model transfer method to resolve the issue of feature space alignment and use the replication padding trick to reduce convolution errors in a distributed environment.
3. Study the transferability of different layers in the model. We demonstrate by experiments how to split a CNN model such that the produced features representation is in a better generalization.

The paper is organized as follows. After an overview of related works in section 2, we present details of AdptVFedConv in section 3. Experimental setup and results are described in section 4. We finally make a conclusion of this paper and discuss about future works in section 5.

## 2 Related Works

### 2.1 Convolution Neural Networks

The convolution neural network, short as CNN, is a type of deep neural network. Unlike the traditional full-connected models, each layer of the CNN consists of a rectangular 3D grid of neurons. The neurons of a layer are only connected to the neurons in a receptive field, which is a small region in the immediately preceding layer, rather than the entire set of neurons. By using receptive fields, CNNs exploit the spatially-local correlation of input data.

VGG[3] is a classical CNN model. A convolution block in VGG consists of a convolution layer, a Batch Normalization layer, and a ReLU activation function. The kernel size of the convolution layer is often set to  $(3 \times 3)$ . By stacking two  $(3 \times 3)$

convolution layers, the model can achieve comparative performance with one ( $5 \times 5$ ) convolution layer, but the number of learnable parameters is smaller. After several convolution layers, there is a max-pooling layer to reduce the size of the input image. An input image will be finally flattened into a vector representation, which is used to predict the true label by full-connected layers.

## 2.2 Federated Learning

Federated learning is a decentralized learning approach that enables multiple participants to collaboratively train machine learning models while keeping the training data on local devices. Existing works on federated learning, such as FedAvg [17], FedOpt [18], and FedMA [19], mainly focus on the horizontal settings. These methods share model parameters during training and aggregate them at the server-side. FedGKT[20] applies the horizontal FL setting but works differently by exchanging hidden features representations. They also introduce the knowledge distillation technique into FL. FedGKT has several advantages compared with conventional FL algorithms such as it requires less (i) edge computation, (ii) communication, and (iii) asynchronous training.

There is no paradigm for vertical federated learning and each conventional machine learning algorithm has a unique version for vertical federated learning scenario. Existing works [6–8, 21] designed their algorithms based on tree models. Work [22] proposed a privacy-preserving SVM classifier over vertically partitioned data. Work in [5] proposes a platform for distributed features by gathering local outputs into a final one using nonlinear and linear transformations. Nonetheless, none of these methods are applicable to CNN models.

## 2.3 Transfer Learning

Fine-tuning is a popular technique which can accelerate training and transfer knowledge from pre-trained model. The transferability of different layers in a neural network raise our attention when designing the algorithm. The researchers in [23] discuss the transferability of various layers in CNNs. The work presented in [24] focus on the transferability of RNNs in natural language processing applications. These two works reached similar conclusions that initializing a network with pre-trained model parameters can provide support for generalization whilst deep layers are not adequate for a model transfer. This paper discusses and evaluates scenarios through which we found the ideal model for transferability of features in deep neural networks.

# 3 Proposed Framework

## 3.1 Problem Setup

In the scenario of vertical FL, datasets share the same sample space but different feature space. Each image is split into several pieces, and different participants own different pieces, i.e., the number of participants and image pieces are equal. In all participants, we choose one, called positive participant, to hold the label data and cannot share the label with others during training. The positive participant also plays

**Table 1:** Summary of symbols

Symbols	illustration
$D^k$	dataset stored in the $k_{th}$ client
$x_i^k$	the $i_{th}$ input image of the $k_{th}$ client
$H_i^k$	hidden representations of input $x_i^k$
$y_i$	labels of input $x_i$
$W_e^k$	feature extractor in the $k_{th}$ client
$W_s$	classifier in the server
$l_{ce}(\cdot)$	cross-entropy loss
$l_{mse}(\cdot)$	mean square error loss

the role of server while other participants play like the client. In this paper, we call the positive participant the server and other participants clients to distinguish their role.

Assuming that there are  $K$  participants and the positive participant is the 1st.  $D$  represents the entire dataset, with the sample number of  $N$  and category number of  $C$ . Let  $X_i^k$  denote the piece of the  $i$ th sample held by participant  $k$ . For server, it owns dataset  $D^1 = \{(X_i^1, y_i)\}_{i=1}^N$  where  $y_i$  is the corresponding label of sample  $X_i$ ,  $y_i \in \{1, 2, \dots, C\}$ . For clients,  $D^k = \{X_i^k\}_{i=1}^N$ ,  $k \in \{2, \dots, K\}$ .

A complete CNN model  $W$  is split into two parts: a feature extractor  $W_e$  and a server classifier  $W_s$ . Each participant creates a local feature extractor  $W_e^k$ . They are similar but not the same, and all initialized with  $W_e^1$ . Each client also needs to train a generator  $W_g^k$  to fine-tune the  $W_e^k$ . The server creates two classifiers, a smaller one  $W_{clst}$  used in the base feature extractor training step and a larger one  $W_s$  to classify aggregated hidden feature maps.

In the prediction step, clients need to send their hidden feature maps  $H_i^k$ , the output of  $W_e^k$ , to the server. The server will aggregate  $H_i^k$  by position information to get  $H_i$  and take it as the input of  $W_s$  to predict the correct label.

The symbols used in the algorithm are summarized in Table 1.

## 3.2 Vertical Federated Learning for CNNs

There are three steps in our proposed approach. In the first step, the server trains a feature extractor through supervised training and broadcasts it to all clients. Second, each client fine-tunes the received model with local data using the architecture of an auto-encoder. Then they calculate the feature representations of local data and upload them to the server. Finally, the server collects extracted feature representations and concatenates them. The server then trains a classifier to predict the true label. The implementation steps of the proposed framework will be discussed in the below sections.

### 3.2.1 Server Pre-training

The first problem we need to solve is how to get a local feature extractor with high generalization. The hidden feature maps extracted by different clients should be structurally similar and aligned so that the server classifier can take them as input. If all clients train their own feature extractors with local data, extracted features will

**Algorithm 1** AdptVFedConv**Input:** Datasets  $D^1, D^2, \dots, D^K$ **Output:** Distributed models  $W_e^1, W_e^2, \dots, W_e^K$ , server classifier  $W_s$ 


---

```

1: ServerTrain():
2: for  $(x_i^1, y_i) \in D^1$  do
3:   // train base feature extractor
4:    $W_e^1 \leftarrow W_e^1 - \eta * \nabla l_{ce}(x_i^1, y_i; w)$ ;
5: end for
6: for each client  $k=2,3,\dots,K$  do
7:   // broadcast  $W_e^1$  and receive returned feature maps
8:    $H^k \leftarrow \text{ClientTrain}(k, W_e^1)$ ;
9: end for
10:  $H \leftarrow \text{concatenate}(H^1, H^2, \dots, H^K)$ 
11: for  $(h_i, y_i) \in (H, Y)$  do
12:   // train server classifier  $W_s$  with feature map  $H$ 
13:    $W_s \leftarrow W_s - \eta * \nabla l_{ce}(h_i^1, y_i; w)$ ;
14: end for
15: return  $W_s$ 
16:
17: ClientTrain( $k, W$ ):
18: Initialize  $W_e^k$  with  $W_e^1$ 
19: for  $x_i^k \in D^k$  do
20:    $W_e^k \leftarrow W_e^k - \eta * \nabla l_{mse}(x_i^k; w)$ ;
21: end for
22:  $H^k \leftarrow f_c^k(W_e^k, X^k)$ 
23: return  $H^k$  to server

```

---

not be aligned in channels. To solve this problem, we use the method of model-based transfer.

It is expected that all clients train their local model with the same initialized model parameters. Therefore, we need one to broadcast its local feature extractor to others. Generally, supervised learning provides better performance compared to unsupervised learning. So, we let the server play this role. The server first trains a complete CNN model with local data. Several shallow layers, denoted as  $W_e^1$ , of the model will be treated as a shared feature extractor to initialize other clients' local models. The rest of the model, denoted as  $W_{clst}$  will be dropped and not be used in the prediction step. It should be noted that there are a few structural distinctions between  $W_{clst}$  and  $W_s$ . Let  $f_p$  represents the pre-training model,  $l_{ce}$  refers to cross-entropy loss function. The aim of this step is optimizing this objective:

$$\arg \min_{(W_e^1, W_{clst})} \sum_{i=1}^N l_{ce}(f_p((W_e^1, W_{clst}); X_i^1), y_i) \quad (1)$$

### 3.2.2 Model Transfer

When clients get the base feature extractor from the server, they train a local autoencoder and fine-tune the feature extractor with local data. An autoencoder is a type of neural network used to learn efficient representations of unlabeled data. The autoencoder works by attempting to reconstruct the input from the hidden representation. The purpose of fine-tuning is to improve the performance of feature extractors in extracting local features. Furthermore, making a difference among different client models is more resistant to hidden vector reconstruction attacks.

Client  $k$  initializes its feature extractor  $W_e^k$  with  $W_e^1$ ,  $k \in \{2, 3, \dots, K\}$ :

$$W_e^k \leftarrow W_e^1 \quad (2)$$

The  $W_e^k$  works as an encoder. We then construct a decoder  $W_g^k$  with inversion structure to the encoder to form an autoencoder. To fine-tune the  $W_e^k$ , the learning rate of  $W_e^k$  is small while that of  $W_g^k$  is normal. By reducing the loss between input image and generated image, the feature extractor adapts to local data distribution. The optimization of the following objective represents the main goal of this step:

$$\arg \min_{(W_e^k, W_g^k)} \sum_{i=1}^N l_{mse}(f_c^k((W_e^k, W_g^k); X_i^k), X_i^k) \quad (3)$$

where  $f_c^k$  represents the autoencoder model held by client  $k$ ,  $l_{mse}$  represents mean-square loss function.

### 3.2.3 Feature Map Aggregation

After getting a local feature extractor, each client calculate representations of local data and updates them to the server. As shown in Figure 1, the server will concatenate these feature representations according to the position:

$$H_i = \text{concatenate}(H_i^1, H_i^2, \dots, H_i^K) \quad (4)$$

Here  $H_i^k = f_c^k(W_e^k, X_i^k)$  means representations uploaded by clients. We assume that the position information of all image pieces is known and fixed.

The server necessitates training a classifier with adjacent feature maps as an input to predict the class of the input. The optimization of the following objective represents the main goal of this step:

$$\arg \min_{W_s} \sum_{i=1}^N l_{ce}(f_s(W_s; H_i), y_i) \quad (5)$$

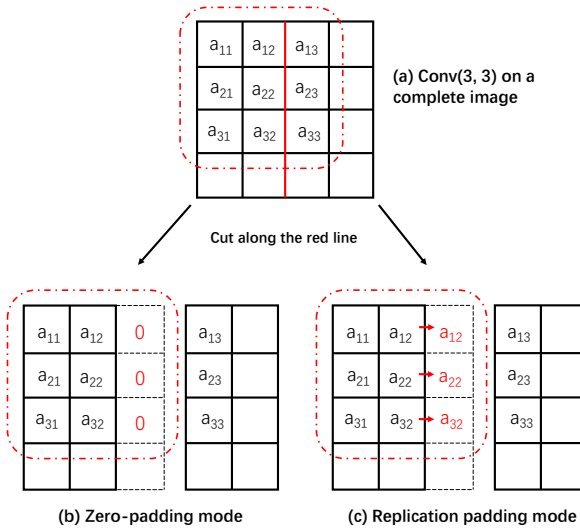
where  $f_s$  represents the server model.

The pseudo-code of all steps is shown in Algorithm 1.



### 3.3 Cause of Accuracy Loss and Optimization

Since FL algorithms work in the distributed scenario and the communication of raw data is forbidden, a pixel can only get information locally. Compared with the centralized scenario, less information makes it hard to train a good feature extractor. We observe in preliminary experiments that there is an accuracy loss in distributed scenarios compared with centralized scenarios. Analyzing the training process of convolution in the two scenarios, we find that when calculating the convolution value of points located on the cutting boundary, the value of some adjacent points cannot be obtained anymore. The image cutting makes these points no longer adjacent and weakens the ability of convolution layers to collect information from surrounding points. The accuracy loss caused by this problem is hard to remove under the condition that exchanging raw data is forbidden. However, by adjusting the padding value to make it closer to the original value, the accuracy loss can be reduced.



**Fig. 2:** Convolution on Cutting Boundary. (a) shows the normal convolution calculation on a complete image. When cutting along the red line, the adjacent point pairs ( $a_{i2}$ ,  $a_{i3}$ ) are assigned to different pieces. (b) and (c) respectively show how the two padding mode works. Replication padding mode is to take the value of the nearest point as the padding value.

In the procedure of calculating convolution values, padding is a common practice to keep the image size while zero-padding is the default mode. As shown in Figure 2, assuming the convolution kernel is fixed, when calculating the convolution value of points on the cutting boundary using the zero-padding mode, the result differs in the two scenarios because the padding value is far different from the original. Empirically, the pixel value of two adjacent points is closer which inspired us to use the replication padding mode to reduce error. Later in Section 4, we will prove the correctness of our analysis by experiment.

### 3.4 Security Discussion

In our work, we let the server share their feature extractor as a base model, and other clients fine-tune it with local data. Note that, the server is also the owner of labels, which collects hidden feature maps from other clients without sharing its own. This means none of the participants share the parameters of their own feature extractor and hidden feature maps simultaneously, which is necessary for protecting the privacy of training data.

On the other hand, a sharing feature extractor is quite necessary for the algorithm. An input image can be separated into three dimensions: RGB channels, height, and width. For example, a sample in the CIFAR-10 dataset has 3 RGB channels, with height and width set to 32 pixels. We denote the size of the input image as  $(3 \times 32 \times 32)$ . A convolution layer is to calculate new feature maps and expand the channels. A max-pooling layer can upsample the value in a small region and reduce the size of the input. Several convolution blocks, which are composed of convolution layers and max-pooling layers, will finally flatten an image into a vector. In this process, each channel represents a unique feature representation. If each client train their local feature extractor without communication, the feature representation is of high possible not aligned. By model transferring and local fine-tuning, each client owns a similar feature and can avoid the problem of feature align.

The number of convolution layers in the local feature extractor is one of the focuses of our study. As shown in Section 4.3, in a convolution neural network, hidden features transition from general to specific by increasing of convolution layers. Fewer layers in the local feature extractor can bring better performance while the security degrades. Therefore, we need to achieve a balance between model performance and data security.

The hidden vector reconstruction attack is a potential threat to our model. Empirically, hidden features after the max pool layer are more resistant to attack than before because the amount of data is halved. In our standard experiment, we take outputs of the first max pool layer as hidden feature maps. More research on the security of exchanging hidden feature maps is needed and we believe that existing methods such as differential privacy and secure multi-party computation can protect data security from the hidden vector attack.

## 4 Experiments

### 4.1 Experiments Structure

#### *Task and Dataset*

The training task of image classification. We perform experiments on three different datasets, CIFAR-10, CIFAR-100, and CINIC-10, which is common in FL research. During generating data for each client, we split a complete input image into four small pieces and assign them to four different clients. All image pieces owned by one participant locate in a fixed position and are the same size. For various methods, we record the top 1 test accuracy as the metric to evaluate the performance of the model.

**Model Architecture**

We employ the architecture of the VGG-16 net to design our model. We divide the VGG-16 net into two parts. A shallow model which consists of several convolution blocks works as the feature extractor. A deep model which consists of both convolution blocks and full-connected layers works as the classifier. Each client trains a local feature extractor and shares representations of local data with the server. The server classifier is trained with adjacent hidden feature representations as input and ground truth labels as the target. In the experiment, as shown in Figure 1, we take the first two convolution layers with the first max pool layer as the feature extractor. The rest of the VGG-16 model serves as the server classifier. The size of the feature extractor and server classifier is adjustable. We can increase the number of layers in the feature extractor and reduce the number of layers in the server classifier synchronously. This may lead to a change in the transferability of feature maps. We will later design more experiments to study it in the ablation study.

In our experiment, the size of the convolution kernel always sets to  $(3 \times 3)$  to capture the information of adjacent pixels and facilitate optimization. For local feature extractors, the input is an image piece of size  $(3 \times 16 \times 16)$  and the output is feature maps having a size of  $(64 \times 8 \times 8)$ . The server will collect feature maps and concatenate them to a complete feature map having a size of  $(64 \times 16 \times 16)$ . For server classifier, the size of hidden feature maps changes in turns:  $(64 \times 16 \times 16) \rightarrow (128 \times 8 \times 8) \rightarrow (256 \times 4 \times 4) \rightarrow (512 \times 2 \times 2) \rightarrow (512 \times 1 \times 1)$ . As the size of an input image decreases, the number of channels increases. An input image is finally converted into a vector of dimension 512. Then, we apply three fully connected layers of dimensions 4096, 4096, and 10 respectively to get the output. In the fully connected layers, we set the dropout rate to 0.5.

When fine-tuning with local data, an auxiliary generative model is needed. The generative model is designed to reconstruct the input, so it has the opposite structure to the feature extractor. For a (Conv, BN, ReLU) block in the feature extractor, there is a (ConvTrans, BN, ReLU) block corresponding to it in the generative model. ConvTrans is the reverse operation of Conv, recovering the input image from its hidden representation. Similarly, we use the upsampling layer to correspond to the max pool layer. The local feature extractor and the generative model work together to form an autoencoder. The feature extractor only needs fine-tuning as mentioned before, so the learning rate of the feature extractor is set to  $1e-5$  while the learning rate of the generative model is set to  $1e-3$ . The client can learn better feature representations by closing the distance between input and generated image.

We also use simulated annealing, image flipping, and random crop to improve the performance, which is the same in all experiments.

**Baseline.**

VFL is a distributed machine learning paradigm for the scenario where features are separated across decentralized clients. For each conventional machine learning algorithm (or model), there is a VFL version. To the best of our knowledge, no existing works study the problem of how CNN models work with vertically partitioned datasets.

Methods	CIFAR-10	CIFAR-100	CINIC-10
<b>AdptVFedConv(ours)</b>	<b>91.30</b>	<b>69.72</b>	<b>80.69</b>
Centralized(VGG-16)	94.04	73.24	84.73
Non-cooperation	65.85	37.88	53.75
No model transfer	87.73	63.87	76.65
Zero-padding mode	90.79	67.55	79.69

**Table 2:** Test Accuracy of Different Methods on Three Datasets

The centralized scenario and the non-cooperation scenario are used as comparison groups in our experiments. In the non-cooperation scenario, we train the model only with local data. We prove that collaborative training can effectively improve model performance when clients do not have sufficient local data. In the centralized scenario, we train a VGG-16 net with a complete dataset as input, meaning that transmitting raw data is allowed. This is the upper bound of our experiments.

To further study the influence of key factors on the experimental results, we design several ablation experiments:

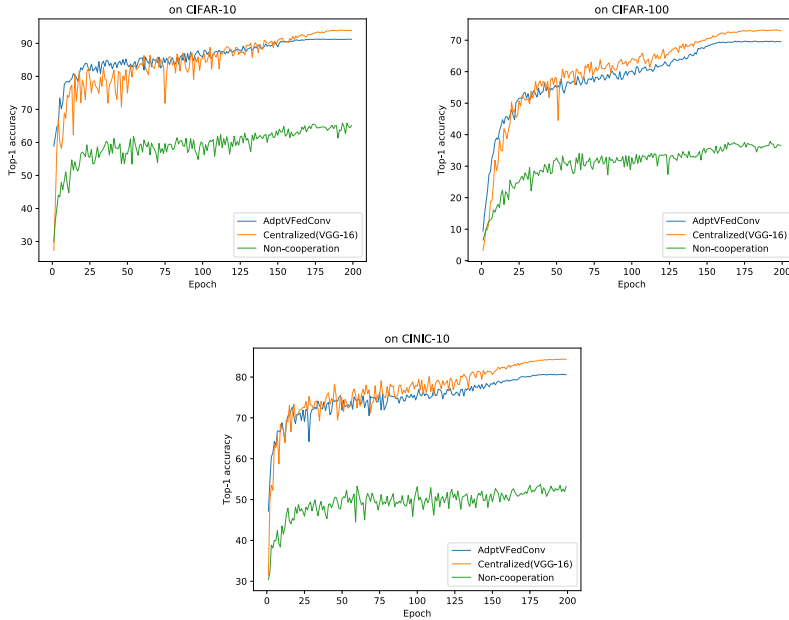
- 1) The first group of ablation experiments aims to study the effectiveness of model transfer. We consider the situation that all clients train their own feature extractor with local dataset independently, i.e., cancel the step (2) in Figure 1.
- 2) The second group of ablation experiments aims to verify the optimization method proposed in section 3.3. We record the top 1 accuracy of two different padding modes, zero-padding mode and replication padding mode.
- 3) The third group of ablation experiments studies the transferability of different layers. In deep neural networks, the hidden representation of different layers carries different information. The output of shallow layers is more similar to the raw input and may leak more information while can train a model for better performance. We want to achieve a trade-off between security and performance through this group of experiments. We adjust the number of convolution blocks in the feature extractor and classifier. Adding a convolution block to the feature extractor means removing one from the classifier and vice versa.

## 4.2 Results Discussion

Figure 3 illustrates the accuracy of the conducted test on three datasets. It includes the result of AdptVFedConv, the centralized model, and the non-cooperation model. We also list all achieved results in Table 2.

It can easily be seen that compared with the non-cooperation model, collaborative training can hugely enhance accuracy of prediction. If we check the image pieces held by each participant, we can find that the information contained in many image pieces is not enough to support predicting the label. Some image pieces even contain only the background. AdptVFedConv can help concatenate decentralized hidden feature maps to form a complete feature representation. It can be seen from the comparison of AdptVFedConv and Non-cooperation curves in Figure 3 that AdptVFedConv improves about 25% to 31% test accuracy on three different datasets.

## Adaptive Vertical Federated Learning via Feature Map Transferring in Mobile Edge Computing



**Fig. 3:** Test Accuracy on Three Datasets.

Compared with the centralized model, the proposed AdptVFedConv achieves very close performance. As analyzed in section 3.3, in the convolution process, accuracy loss is unavoidable since the edge points cannot take values from the adjacent pixels normally. The accuracy of the centralized model is the threshold of our experiment since exchanging raw data is allowed. This work is to make the performance of the distributed model closer to the upper bound. The amount of accuracy loss is related to the number of Conv layers in the feature extractor. In the ablation study, we will demonstrate that the test accuracy degrades as the number of Conv layers in the feature extractor increases.

### 4.3 Ablation Study

We design several ablation experiments to verify the effectiveness of key factors on experiment results.

#### *Effectiveness of Model Transfer.*

As mentioned in the above sections, all clients fine-tune a shared feature extractor locally. In this group of ablation experiments, we prove that starting training with a model initialized with the same parameters can boost model performance. We omitted step 1 in Figure 1, making all clients train their local feature extractor with local data independently. Since most participants have no label data, they can only use an autoencoder to train the feature extractor. The row of 'No model transfer'

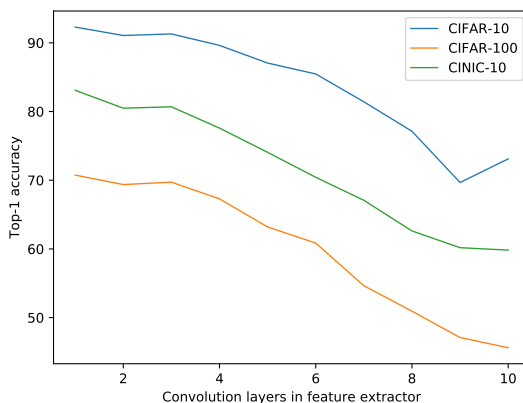
Layers	CIFAR-10	CIFAR-100	CINIC-10
1C	92.30	70.75	83.10
2C	91.07	69.37	80.49
2C+1M	91.30	69.72	80.69
3C+1M	89.63	67.28	77.59
4C+1M	87.06	63.20	74.06
4C+2M	85.47	60.85	70.43
5C+2M	81.42	54.64	67.08
6C+2M	77.13	50.94	62.62
7C+2M	69.67	47.10	60.18
7C+3M	73.11	45.62	59.83

**Table 3:** Accuracy vs Layer Numbers in the Feature Extractor. The first column represents the composition of the feature extractor. 1C represents a convolution block while 1M represents a max-pooling layer. On the whole, the accuracy decreases with the increase of layer numbers.

in Table 2 demonstrates that, without model transferring, the test accuracy degrades significantly. We can see that model transfer is of great help to train a better model.

### *Effectiveness of the Replication Padding Trick.*

Table 2 shows the results of the efficacy of using replication padding mode. We record the top 1 accuracy in the scenario that all participants using zero-padding mode as a comparison. We observe that the accuracy of zero-padding mode is always lower than that of replication padding mode, which proves the effectiveness of our optimization method. This group of experiments also prove the correctness of our analysis on accuracy loss. To eliminate the accuracy loss, it is necessary to exchange the values of edge points in each convolution calculation step, which is forbidden in our setting.



**Fig. 4:** Accuracy Test for Various Layer Numbers in the Feature Extractor.

***Transferability of Different Layers.***

We change the splitting position of the VGG-16 model and record the top 1 accuracy of each splitting position. We take a complete convolution process, including a convolution layer, a batch normalization function, and an active function, as a base convolution unit. We also record the effect of the max pool layer, since it can effectively reduce the communication cost and make the communication more secure. From Table 3, we foresee that there is an indirect relationship between accuracy and layer numbers in the feature extractor: accuracy decreases whilst layer numbers increases. This is because of features transition from general to specific by the deepening of convolution. Note that, in all three datasets, adding the first max pool layer into the local feature extractor has no negative impact on the accuracy. This is of help for us to achieve a balance between performance and security.

## 5 Conclusion and Future Work

In this work, we presented a new image classification algorithm through a vertical FL scenario. Unlike the conventional algorithms for sharing model parameters, the proposed algorithm enabled sharing hidden feature representations which is more similar to human behavior. AdptVFedConv is the first vertical FL algorithm to support CNNs and introduced multiple advantages over other state-of-the-art approaches such as less demand for edge computation and communication cost. Although there is an accuracy loss compared with the centralized model, we proposed several optimization methods to reduce it and the experiments proved the efficacy.

In this study, image pieces owned by one participant are always located in a fixed position and are all the same size. This follows the scenario of vertical federated learning but is too idealistic. In future research, we plan to relax the restriction to make our method more practical.

## 6 Acknowledgments

This work was supported by the National Key Research and Development Program of China under Grant 2020YFB1712101, National Natural Science Foundation of China (no. U1936218 and 62072037).

## References

- [1] Yang Qiang, Liu Yang, Chen Tianjian, and Tong Yongxin. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.
- [2] Wenliang Du and Mikhail J Atallah. Privacy-preserving cooperative statistical analysis. In *Seventeenth Annual Computer Security Applications Conference*, pages 102–110. IEEE, 2001.

- [3] Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition *International Conference on Learning Representation*, 2015.
- [4] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Secure linear regression on vertically partitioned datasets. *IACR Cryptol. ePrint Arch.*, 2016:892, 2016.
- [5] Yaochen Hu, Di Niu, Jianming Yang, and Shengping Zhou. Fdml: A collaborative machine learning framework for distributed features. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2232–2240, 2019.
- [6] Jaideep Vaidya and Chris Clifton. Privacy-preserving decision trees over vertically partitioned data. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 139–152, 2005.
- [7] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems* 36.6, 2021.
- [8] Yang Liu, Yingting Liu, Zhijie Liu, Yuxuan Liang, Chuishi Meng, Junbo Zhang, and Yu Zheng. Federated forest. *IEEE Transactions on Big Data*, 2020.
- [9] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 619–636, 2016.
- [10] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine learning models that remember too much. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 587–601, 2017.
- [11] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2017.
- [12] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–618, 2017.
- [13] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pages 14774–14784, 2019.
- [14] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients—how easy is it to break privacy in federated learning?



*Adaptive Vertical Federated Learning via Feature Map Transferring in Mobile Edge Computing*

*Advances in Neural Information Processing Systems*, pages 16937-16947, 2020.

- [15] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.
- [16] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.
- [17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.
- [18] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *International Conference on Learning Representations*, 2020.
- [19] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. *International Conference on Learning Representations*, 2019.
- [20] Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
- [21] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. Privacy preserving vertical federated learning for tree-based models. *Proc. VLDB Endow.*, 13(11):2090–2103, 2020.
- [22] Hwanjo Yu, Jaideep Vaidya, and Xiaoqian Jiang. Privacy-preserving svm classification on vertically partitioned data. In *Pacific-asia conference on knowledge discovery and data mining*, pages 647–656, 2006.
- [23] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.
- [24] Lili Mou, Zhao Meng, Rui Yan, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. How transferable are neural networks in nlp applications? *EMNLP*, 2016.