2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)

A Lightweight Collaborative Recognition System with Binary Convolutional Neural Network for Mobile Web Augmented Reality

Yakun Huang
Beijing University of Posts and Telecommunications
Beijing, China
Email: ykhuang@bupt.edu.cn

Xiuquan Qiao
Beijing University of Posts and Telecommunications
Beijing, China
Email: qiaoxq@bupt.edu.cn

Pei Ren
Beijing University of Posts and Telecommunications
Beijing, China
Email: renpei@bupt.edu.cn

Ling Liu
Georgia Institute of Technology
GA, USA
Email: lingliu@cc.gatech.edu

Calton Pu
Georgia Institute of Technology
GA, USA
Email: calton.pu@cc.gatech.edu

Junliang Chen
Beijing University of Posts and Telecommunications
Beijing, China
Email: chjl@bupt.edu.cn

***Abstract*—Lightweight and precise recognition is a key component of web-based augmented reality (Web AR) applications. Although edge-based distributed deep learning approach is now possible to achieve satisfactory recognition for Web AR applications, it puts significant pressure on the computation and energy consumption of the mobile web browser, especially the app-based embedded browser. Thus, reducing the model size and accelerating the inference are regarded as the two fundamental challenges to enable this edge-based collaborative recognition system efficiently. In this paper, we propose a lightweight collaborative recognition system (LCRS) for Web AR applications. LCRS contributes to three aspects: (1) we design a composite deep neural network for reducing the model size and inference latency by introducing binary convolutional neural network; (2) we provide a joint training method to co-train the general branch and the binary branch; (3) we develop a JavaScript library for the mobile web browser to execute and accelerate inference of the binary branch, which also provides a collaborative mechanism between the mobile web browser and the edge server. We have conducted extensive experiments using several well-known networks and datasets. The experimental results have shown that the proposed system outperforms the existing approaches in terms of reducing the model size by about 16x to 29x, and it also reduces end-to-end latency and outpaces the existing state-of-the-art approaches by over 3x to 60x when applying it in practical Web AR cases.**

***Keywords*-Distributed deep neural networks, binary convolutional neural network, mobile computing, edge computing, web augmented reality.**

I. Introduction

Recently, the augmented reality (AR), a supplying users' perception of the real world, can benefit users in extensive application fields, and the potential uses of AR are increasing. However, both wearable-device-based and app-based solutions suffer from cross-platform, portability, and cost issues. Web-based AR, a lightweight and cross-platform approach, is promising for mobile users to interact with real-world [1], [2]. Object recognition, as the primary step of Web AR applications, is crucial because incorrect recognition will result in an ineffectual tracking and rendering. Traditional algorithms such as SIFT [3], SURF [4] and ASIFT [5] provide fast feature extraction and matching while the generalization capacity is insufficient. Due to stronger recognition capacity of Deep Neural Networks (DNNs), web-based technologies and various executing schemes leveraging DNNs into the mobile web browser have become a popular research topic.

The frequently used deep-learning based recognition approaches for the mobile Web AR such as mobile-only and edge-only, are confronted with unsatisfactory practice [6], [7]. Mobile-only approaches execute the entire DNN inference on the mobile web browser without network communication costs. Although JavaScript-based DNN frameworks such as Keras.js [8] and Tensorflow.js [9] provide common neural layers to run trained models on the web browser, limited resource of the mobile web browser, especially the app-based embedded browser makes an unbearable latency facing with large models and massive computation (e.g., the model size of AlexNet is up to 249MB with 61M parameters). Considering the limitations of terminal capabilities, edge-only approaches offload the whole task to the edge server and return recognized results to the mobile web browser. However, expensive computing costs of the edge server discourage service providers to offload the entire computation to the edge servers in practical scenarios (e.g., the computing cost of high concurrent requests is unacceptable). Moreover, the transmission latency of larger images is also intolerable in an unstable wireless communication environment.

Unlike these approaches, it is natural to consider the

2575-8411/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDCS.2019.00149

use of a distributed computing approach to alleviate the computation by partitioning model and offloading partial computations from the edge server to the mobile web browser. For example, aiming at mobile devices rather the mobile web browser, Neurosurgeon [10] tries to acquire minimum communication and sufficient resource usage of mobile devices by deploying partitioned DNN model on mobile devices in advance. This approach has the benefit of low communication costs and can offload proper computation to the mobile devices compared to the edge-only approach.

However, employing this kind of partition-offloading approach onto the mobile web browser is challenging for three reasons: (1) It is impracticable to deploy partitioned models onto the mobile web browser in advance as web pages are loaded on demand in real time. Thus, the partition-offloading approach will suffer from the communication costs of loading model. (2) The latency of loading partial model and inference is still too high for the mobile web browser regardless of how the deep neural network is partitioned (e.g., the size of the first convolutional layer of AlexNet over ImageNet reaches 14MB). (3) Although WebDNN [11] and ONNX.js [12] leverage WebGL [13] and WebAssembly [14] technologies for providing an optimized model inference, they are short of supporting partition-offloading approach between the mobile web browser and the edge server.

To address these concerns, aiming at executing lightweight recognition on the mobile web browser while maintaining accuracy, we first design a composite deep neural network by introducing a binary convolutional neural network as the side branch to acquire the lightweight recognition. Because there is no available training method for proposed composite network, it is necessary to jointly train the binary branch with the general neural network branch. The aforementioned JavaScript libraries such as Keras.js, Tensorflow.js and ONNX.js, are implemented for general neural networks and not aimed at binary convolutional networks. Hence, we further implement an inference library with a collaborative paradigm for the binary branch on the mobile web browser. The main contributions of this paper are as follows:

- **A composite architecture of lightweight collaborative deep neural network** - We design a lightweight collaborative recognition system of the deep neural network with binary convolutional network aiming to reduce the unbearable latency of loading model and inference on the mobile web browser. We further leverage the collaboration of the main branch deployed at the edge server to compensate for the recognition shortage of binary branch.
- **A joint training method with early exit binary branch** - A joint training method consists of training the main branch, binarizing inputs and weight filters and training binary branch, which allows low latency recognition for simple inputs via early exit of a binary branch but with acceptable accuracy.
- **Fast inference library of the binary branch for web-based collaborative DNN** - We implement an inference library for executing trained binary branch on the mobile web browser and collaborating with the edge server. To our best knowledge, this library firstly provides the capability to run inference of binary deep neural network on the mobile web browser.

II. Related work

A. Distributed inference of deep neural networks

Partition-offloading is the most fundamental methodology to execute distributed inference working on traditional deep neural networks. Recently, Neurosurgeon automatically chooses the partition points pursuing the optimal latency and energy consumption to offload DNN computation. Edgent [15] searches the adaptive partitions of DNN computation and accelerates DNN inference through an early exit at a proper intermediate DNN layer. Similarly, there are prior explorations focusing on intelligent collaboration between the mobile device and the cloud without combining the granularity of neural network layers. McDNN [16] investigates the intelligent collaboration to execute either in the cloud or on the mobile device by generating alternative DNN models for performance and energy costs. MoDNN [17] proposes two partition schemes to minimize non-parallel data delivery latency and accelerate DNN inference by alleviating device-level computing cost and memory usage. Moreover, JointDNN [18] further proposes efficient DNN inference giving the training process simultaneously, which also provides various optimizations in DNNs tackling with resource constraints.

However, these anterior researches only benefit the lightweight DNN models, and they do not support deeper neural networks (e.g., AlexNet and VGG16) because of massive weights and computations. Thus, it is worthy of considering lightweight neural networks like binary convolutional neural network for reducing the communication and computation costs.

B. Binary neural network

As one of the effective compression methods, binary convolutional neural network attempts to address efficient training and inference by binarizing weights and activations compared to typical deep neural networks. The directions are of two kinds: Expectation BackPropagation (EBP) and BinaryConnect. EBP in [19] achieves a neural network with binary weights and binary activations by variational Bayesian approach. Esser et al. [20] treats spikes and discrete synapses as continuous probabilities, which allows to train the network using standard backpropagation and shows the advantages in energy efficiency. BinaryConnect [21] trains a DNN with binary weights during the forward and backward propagation while retaining precision of stored

weight. It shows high performance on small datasets while large-scale datasets are not suitable. XNOR-Network [22] is more extreme to binarize filters and inputs in convolutional layers which results in faster and more memory-saving inference. Undoubtedly, network binarization makes the trade-off between the model size and precision. Nevertheless, it is hard to acquire satisfactory precision with an efficient compression of networks in terms of complex datasets (e.g., ImageNet).

C. Web-based recognition of augmented reality

Image recognition, as a core component of Web AR applications, is generally realized with traditional feature extraction (e.g., SIFT [3]) and deep learning-based algorithms. The generalization capability of deep neural networks offers stronger recognition ability than SIFT-based algorithms when confronted with complex environments. However, web-based deep neural network restricts the hardware of mobile device (e.g., WebGPU) to accelerate inference. CaffeJS [25] loads pre-trained deep neural networks entirely in JavaScript, which aims to execute forward and backward propagation. All of this runs on mobile devices without installing any software. Similarly, Keras.js [8] supports GPU and CPU mode of Keras models in the browser, which can be trained in any backend, including TensorFlow, CNTK, etc. TensorFlow.js [9] is a JavaScript library for training and deploying deep learning models in the browser and on Node.js. WebDNN [11] also provides an installation-free DNN execution environment by optimizing the trained DNN model to compress model data and accelerating the execution.

Although the aforementioned technologies provide the chance to execute the entire DNN inference on the mobile web browser, they get into trouble with higher latency, computation constraints and loss of accuracy. Our approach in this paper provides a lightweight collaborative recognition system to alleviate the conflict between the model size and recognition accuracy.

III. Background and Motivation

A. Distributed inference of deep neural networks

We describe the detail process of web-based DNN recognition between the edge server and the mobile web browser, consisting of offline training, partition and distributed execution in Figure 1. The public cloud provides trained DNN models to cope with complex applications. The edge server loads DNN models from the public cloud and makes decisions for partitioning DNNs models. Mobile web browser executes the partial inference once receiving tasks (e.g., images, media, videos), and transfers intermediate results to the edge server for the rest of DNNs inference. Notice that there may exist several transmissions of intermediate results between the edge server and the mobile web browser. Thus, communication costs introduce an unbearable latency

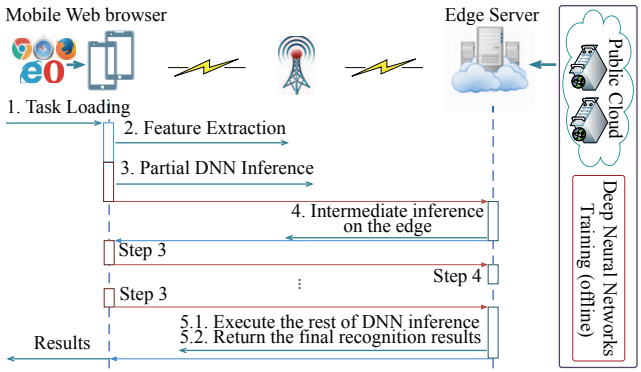

Figure 1. Distributed inference scheme of deep neural networks.

during the entire inference. More importantly, whatever partitioning point is chosen, it is not lightweight enough to execute partial or entire inference on the mobile web browser because of a large model size.

B. Accelerating inference of deep neural networks

There are various methods for accelerating DNN inference such as BranchyNet [26] and binary convolutional neural network [22]. BranchyNet provides early exit points for accelerating inference while binary convolutional neural network dramatically reduces the model size and inference latency by binarizing inputs and weights.

1) BranchyNet: BranchyNet pursues fast inference via early existing from deep neural networks. For a simple BranchyNet with three branches added to the original deep neural network, each branch consists of a convolutional layer and fully connection layer freely. By this character, the exit points allow the majority of tasks to exit early, thus reducing the needless inference. It is useful that BranchyNet uses the entropy of a classification result as a measure of exiting.

2) Binary convolutional neural networks: The filters of binary convolutional neural networks are approximated with binary values and factors resulting in memory-saving. XNOR-Network is more radical to binarize both inputs and filters of convolutional layers. Traditional convolutional operations are replaced by approximate convolutions using primarily binary operations, which provide faster convolutional computation and more memory-saving operations.

C. Key properties of distributed DNN for Web AR

Web AR is coming trendy for its installation-free and interesting attraction, while limited resource hinders its development. However, traditional deep neural networks require a large amount of model and computation, which are too heavy to run on the mobile web browser directly even executing partial DNN models. Thus, in this paper, we point out several key properties of recognition of Web AR applications to guide our approach.

- **Lightweight.** Compared to app-based recognition frameworks, limited resource of the mobile web browser requires a more lightweight framework which should be seamless with the existing technology of the mobile web browser.
- **Low latency.** Low latency generally includes transmission and computation latency. Thus, it requires that the model size of deep neural networks should be suitable for the mobile web browser.
- **A small amount of computation.** Deeper neural networks are trained with large amounts of weights and computations. Thus, the computation of inference must be limited to an appropriate threshold.
- **Installation-free and deployment-free.** Because the majority of excellent deep neural networks are depended on the special server equipped with GPUs, an effective library for transforming traditional DNNs execution to suitable format on the mobile web browser is important progress of Web AR.
- **Collaborative mechanism.** Due to various constraints of the mobile web browser, we consider offloading computations to the mobile web browser confronted with complicated tasks and environments. Therefore, supporting collaborative mechanism should be one requisite property of Web AR applications.

IV. Overview of the Collaborative Recognition System

Adding branches into traditional deep neural networks reminds us to leverage this into distributed inference among heterogeneous devices like the edge server, the mobile device and the ensemble device. Although we design an optimal branch for the mobile device, parameters of partial deep neural networks are still too large to execute in real-time. Moreover, a binary convolutional neural network can reduce memory usage dramatically though it coexists with the loss of accuracy. To address this shortcoming, we propose a lightweight collaborative system via designing a composite deep neural network with a binary convolutional neural network, called LCRS. Because there is no available approach for binary branch inference on the mobile web browser directly. We are first to implement an inference library to transform the original binary convolutional neural network to JavaScript library based on WebAssembly. This library firstly provides the capacity to run binary deep neural networks on the mobile web browser, which also have a collaborative mechanism with the edge server flexibly.

A. The architecture of LCRS

LCRS trains the composite deep neural network for the mobile web browser and the edge server by adding one binary convolutional neural network. This architecture benefits of several aspects, namely promoting the inference efficiency, reducing memory usage and providing a collaborative scheme to supply accuracy of the binary branch.

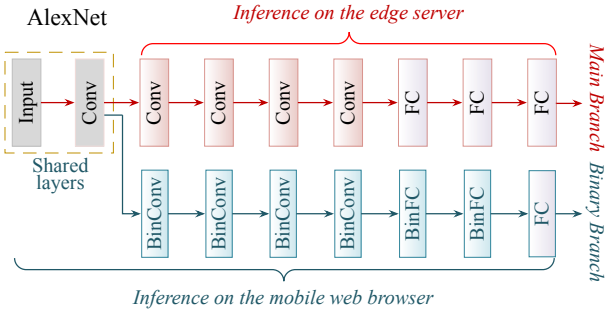

Figure 2. The architecture of LCRS.

We take AlexNet as an example that both the main branch and the binary branch share the first convolutional layer in Figure 2. For one thing, if we binarize all layers in the binary branch, which will result in a dramatic loss of recognition accuracy with few reductions of memory usage. For another, when the recognition results of the binary branch cannot satisfy the tasks, we only need to transfer intermediate outputs to the edge server rather initial tasks. Take advantage of this shared structure, and the edge server can provide the accuracy supplement for the binary branch. We can free the intermediate outputs when the binary branch has confidence for tasks. Otherwise, the mobile web browser frees them after sending them to the edge server.

B. Training joint deep neural network

The loss function of the training process of LCRS consists of the main branch and the binary branch. In the feedforward of the main branch, we calculate training datasets through the network including all branches. Especially, LCRS binarizes both inputs and weights which introduces the alpha factor to approximate with traditional feedforward pass of typical convolutional layer. Meantime, it records the outputs from the exit points of all branches for calculating the error of the network. Then, for backward propagation, gradient descent like Adam algorithm updates the weights by the error passed back through the main branch network. Due to tiny changes of parameters in gradient descent, we only binarize the weights during the forward pass and backward propagation, which uses high precision weights to update parameters and is also employed to train a binary network in [21], [22] and [24].

We use AlexNet as an example for image classification and softmax cross entropy loss function is used as the optimization objective. To train LCRS, we aim to minimize the loss function of all branches that form a joint optimization problem as a sum of the loss functions of the main branch

and the binary branch.

$$\mathcal{L}(\hat{y}, y; \theta) = \mathcal{L}_{main}(\hat{y}_{main}, y; \theta) + \mathcal{L}_{binary}(\hat{y}_{binary}, y; \theta). \tag{1}$$

Where y is a one-hot ground-truth label vector, $\hat{y}$ is the predicted label vector. For the loss function of the main branch, $\mathcal{L}_{main}(\hat{y}_{main}, y; \theta)$ can be represented as

$$\mathcal{L}_{main}(\hat{y}_{main}, y; \theta) = -\frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} y_c \log \hat{y}^c_{main}, \tag{2}$$

where

$$\hat{y}_{main} = \mathrm{softmax}(z) = \frac{e^z}{\sum_{c \in \mathcal{C}} e^z}. \tag{3}$$

Where $z = f_{main}(x; \theta)$ denotes the output of the main branch with the input sample x. $\mathcal{C}$ represents the set of all possible labels. Hence, we are easy to execute feedforward pass and backward propagation with gradient descent.

For the binary branch, we firstly binarize weights of the neural network and estimate binary weights. During the updating of parameters, we use high precision weights preventing gradient disappears of binarization. The forward propagation of the training process is similar to the standard forward propagation except that convolutions are computed as

$$I \times W \approx (\mathrm{sign}(I) \otimes \mathrm{sign}(W)) \odot K \cdot \alpha. \tag{4}$$

The following equation exhibits the core convolutional operation, which approximates the convolution between input I and weight filter W using binary operations. Here, α is the scaling factor for the weight and K denotes all sub-tensors.

During the backward propagation, we follow the same approach as [26] to compute the gradient for $\mathrm{sign}(x)$.

$$\frac{\partial \mathrm{sign}}{\partial x} = x_{1_{|x| \leq 1}} \tag{5}$$

Thus, we calculate the gradient in backward propagation after the scaled sign function is

$$\frac{\partial \mathcal{L}}{\partial W_i} = \frac{\partial \mathcal{L}}{\partial \widetilde{W}_i}(\frac{1}{n} + \frac{\partial \mathrm{sign}}{\partial W_i}\alpha), \tag{6}$$

where the gradients are computed with respect to the estimated weight filters $\widetilde{W}_i$.

Algorithm 1 presents our procedure for training N-Layers LCRS. The whole process consists of training the main branch, binarizing weight filters and training the binary branch. We only perform forward propagation with binarized weights at the inference of the binary branch without keeping the full precision weights.

C. Web-based collaborative DNN inference with efficient forward library

Once the training is finished, LCRS can load and execute inference efficiently because of tiny binary branch. For a given sample x, if the binary branch is confident to predict the result and satisfy users, the sample can exit from the binary branch directly. Otherwise, we have to transfer the output of the first convolutional layer to the edge server for a precise result. Thus, to measure the classifier accuracy of the binary branch, we introduce normalized entropy, $S(x) \in [0, 1]$ for the exit point of the binary branch which is also employed in [23].

$$S(\mathbf{x}) = -\sum_{i=1}^{|\mathcal{C}|} \frac{x_i \log x_i}{\log |\mathcal{C}|} \tag{7}$$

Then, we compared $S(\mathbf{x})$ against a threshold to determine whether or not to exit from the binary branch. In practice, the optimal value of the exit threshold for the binary branch depends on the applications and datasets. BranchyNet [26] provides an approach for performing screening and picking a setting that satisfies the constraints over τ. Similarly, we pick appropriate τ for various networks and datasets in the same way. We also present the collaborative inference between the mobile web browser and the edge server in Algorithm 2.

Note that existing deep learning frameworks are implemented in Python or C++ which do not support the mobile web browser for augmented reality. JavaScript and WebAssembly are the dominant methods for executing in-

Algorithm 1 Training process of a N-Layers LCRS:

Input:
X, Y: a minibatch of inputs and labels;
$\mathcal{L}(\hat{Y}, Y)$: loss function;
W^l_{main}: current layer weights of the main branch;
η^l_{main}: current learning rate of the main branch;
W^l_{binary}: current layer weights of the binary branch;
η^l_{binary}: current learning rate of the binary branch.

Output:
Updated weights $W^{l+1}_{main}, W^{l+1}_{binary}$ and learning rate $\eta^{l+1}_{main}, \eta^{l+1}_{binary}$

Traing main branch:
$\hat{Y}_{main} \Leftarrow StandardForward(X, W_{main})$;
$\frac{\partial \mathcal{L}}{W^l_{main}} \Leftarrow StandardBackward(\frac{\partial \mathcal{L}}{\hat{Y}_{main}}, W^l_{main})$;
$W^{l+1}_{main} \Leftarrow Update(W^l_{main}, \frac{\partial \mathcal{L}}{W^l_{main}}, \eta^l_{main})$;
$\eta^{l+1}_{main} \Leftarrow Update(\eta^l_{main}, l)$;
Traing binary branch:
//Binarizding weight filters from the second convolutionl layer
for $n = 2$ to N **do**
  $\widetilde{W}^l_n \Leftarrow \frac{1}{n} \left\|W^l_n\right\|_{\ell 1} \cdot \mathrm{sign}(W^l_n)$;
end for
$\hat{Y}_{binary} \Leftarrow BinaryForward(X, \mathrm{sign}(W^l), \frac{1}{n} \left\|W^l_n\right\|_{\ell 1})$;
$\frac{\partial \mathcal{L}}{\widetilde{W}^l} \Leftarrow BinaryBackward(\frac{\partial \mathcal{L}}{\hat{Y}_{binary}}, \widetilde{W}^l)$;
$W^{l+1}_{binary} \Leftarrow Update(W^l_{binary}, \frac{\partial \mathcal{L}}{\widetilde{W}^l}, \eta^l_{binary})$;
$\eta^{l+1}_{binary} \Leftarrow Update(\eta^l_{binary}, l)$;
return $W^{l+1}_{main}, \eta^{l+1}_{main}, W^{l+1}_{binary}, \eta^{l+1}_{binary}$;

Algorithm 2 Fast distributed inference of LCRS:

Input:

x: an input sample;

τ: a threshold for determining wether to exit.

Output:

Predicted result

$t \Leftarrow f_{main}^{Conv1}(x)$ // Output of the first convolutional layer

$z_{binary} \Leftarrow f_{binary}(t)$

$\hat{y}_{binary} \Leftarrow \mathrm{softmax}(z_{binary})$

$e \Leftarrow S(\hat{y}_{binary})$

if $e < \tau$ **then**

  return $\arg\max \hat{y}_{binary}$

else

  $z_{main}^{rest} \Leftarrow f_{main}^{rest}(t)$

  $\hat{y}_{main}^{rest} \Leftarrow \mathrm{softmax}(z_{main}^{rest})$

  return $\arg\max \hat{y}_{main}^{rest}$

end if

ference on the mobile web browser. Thus, we implement the LCRS feedforward library in C++ and convert them to JavaScript and WASM to execute on the mobile web browser directly. We present the whole process for executing LCRS on the mobile web browser in Figure 3.

Once trained in Python (e.g., Pytorch), inference scripts implemented in C++ convert the first convolutional layer and binary convolutional layers into JavaScript and WASM by Emscripten [28]. We also validate the correctness of our implementation by comparing the outputs to the inference of Pytorch.

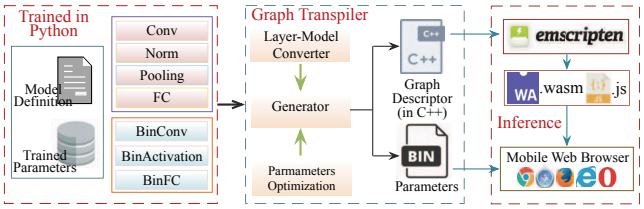

Figure 3. Fast inference library on Web browser

D. Discussions

In this part, we discuss and verify the reasonableness of LCRS architecture from the following aspects.

1) The number of binary branches: Taking adding two binary branches as an example, e_1 and e_2 denote the entry points on the main branch. p_1 and p_2 denote the recognition precision of these two binary branches. Our experiments reveal that the second branch has a little lifting than the first branch that layer distance is close between the two (e.g., Adding the binary branch after the first convolutional layer acquires the largest profit in AlexNet). This means adjacent location of e_2 and e_1 has little promotions between p_2 and p_1 resulting in a large expectation latency because of more communication costs of the second branch. If we increase the layer distance between the first branch and the second branch, though we obtain the increase of accuracy, the full precision mode size of the second branch introduces large communication costs which losts the advantages of adding the binary convolutional networks.

Especially, in a real environment, the network bandwidth is instability resulting in large communication costs during the interactions between the mobile web browser and the edge server. In conclusion, if we add one binary branch to the main branch, there is only an interaction, even the binary branch is not confident of the sample task. Thus, considering the communication costs, size and accuracy lifting of the binary branch, we design a collaborative deep neural network with only one branch which is a binary convolutional neural network to support the real web AR applications.

2) Location of binary branch: Generally, we suggest adding one binary branch after the first convolutional layer. The optimum location of the binary branch is discussed based on the aforementioned definitions. Let $e_1 = 1$ represent adding the binary branch after the first convolutional layer. Thus, for any h, e_h, ($2 < e_h < l$) is the binary branch of the e_h^{th} layer. E_n represents the expectation of inference latency.

We can obtain $E_{e_h} - E_{e_1} > 0$ due to communication costs and a small amount of accuracy lifting. The main reason is that if e_h is close to l, the whole network degenerates into a traditional deep neural network with large weights and computations. When e_h is close to e_1, the accuracy lifting is slight while increasing the communication costs. Thus, if we consider adding only one binary branch, it is preferable to add it after the first convolutional layer.

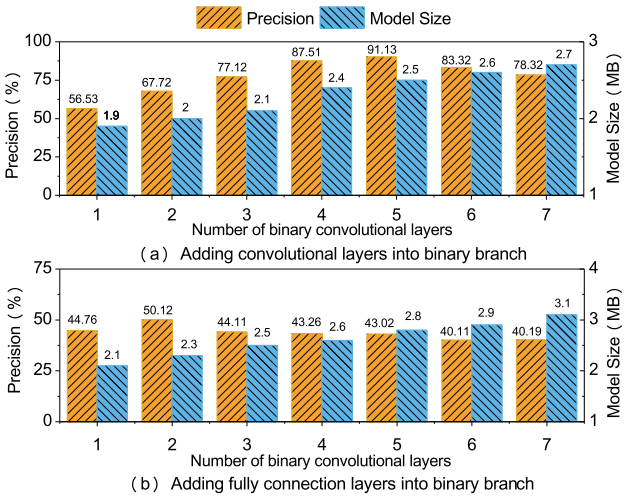

Figure 4. The structure of the binary branch.

3) Structure of binary branch: In this paper, the binary branch mainly consists of binary convolutional layers and

binary full connection layers, and we only consider adding a binary branch after the first convolutional layer based on the above discussions. Using AlexNet network as the main branch, Figure 4 exhibits the performance of accuracy and model size of various structures of binary branches. Concretely, the structures of Figure 4(a) are n, $(n < l)$ binary convolutional layers and a binary fully connection layer exploring the influence of binary convolutional layer. In Figure 4(b), we use only one binary convolutional layer and n, $(n < l)$ binary fully connection layers to reveal the effectiveness in the binary branch. The last layer of all structures is a full connection layer with float weights.

Experimental results show that it is not a better choice to add more binary convolutional layers into the main branch due to the accuracy decrease. Similarly, one or two binary fully connection layers may introduce higher accuracy. Hence, we suggest consulting the structure of the main branch when designing the binary branch for a satisfactory experience.

V. Experiments and applications

We evaluate LCRS using typical deep neural networks of LeNet, AlexNet, ResNet18 [27] and VGG16 over datasets such as MNIST, Fashion-MNIST, CIFAR10 and CIFAR100.

A. Training results of LCRS

Because image sizes are 28*28 or 32*32 of MNIST, FashionMNIST, CIFAR10 and CIFAR100, we adjust several parameters of networks such as input channel and output channel for better training performance. Table I shows the main performance on accuracies of the main branch and the binary branch, exiting threshold, exit possibility, the model size of the main branch and the binary branch respectively. M_Acc and B_Acc represent training accuracies of two branches respectively and exit possibility denotes the ratio of exiting from the binary branch in 100 random samples. We also compare the model size of two branches to highlight the advantage of the binary branch.

On the whole, the binary branch reduces memory usage about 16x to 30x when compared to the main branch. From the aspect of training accuracy, the network with shallow binary layers has little accuracy reduction compared to the main branch. However, the deeper network performs a sharp decrease due to more binary convolutional layers. Hence, we evaluate exit possibility of the binary branch with a baseline of 100 random samples. Concretely, LeNet's binary branch contributes to around 90 percentages of samples exiting without the collaboration of the main branch. Meanwhile, τ is stricter than other networks to acquire more precise recognition. Furthermore, LeNet's binary branch obtains 16 times memory-saving compared to the full precision model. For the second layers of networks, the binary branches' accuracies have obvious decrease because more binary convolutional layers result in a larger loss of precision. The results show that AlexNet's binary branch on CIFAR10 still acquires great percentages of exit possibility with a proximity training accuracy to LeNet. We find that AlexNet' τ is larger than LeNet which reflects the dependence of the network structure. The binary branch is memory-saving while recognition accuracy has serious descent in deeper networks. VGG16 network only has an accuracy of 65 percentages on CIFAR100 when executing the binary branch on the mobile web browser. The exiting probability performs an associated decrease to 76 percentages.

Table I
Performance of training results

Network/ Dataset	M_Acc. (%)	B_Acc. (%)	Threshold (τ)	Exit (%)	M_size (MB)	B_size (MB)
LeNet-MNIST	99.50	98.81	0.0001	94	1.7	0.103
LeNet-FashionMNIST	99.41	98.67	0.0001	93	1.695	0.102
LeNet-CIFAR10	65.49	63.21	0.0001	84	1.71	0.102
LeNet-CIFAR100	55.32	54.23	0.0001	83	1.7	0.103
AlexNet-MNIST	97.26	95.34	0.025	87	90.906	3.3
AlexNet-FashionMNIST	97.89	96.12	0.025	87	90.905	3.3
AlexNet-CIFAR10	76.85	73.99	0.025	79	90.911	3.3
AlexNet-CIFAR100	57.31	54.73	0.025	76	92.351	3.5
ResNet18-MNIST	97.91	96.13	0.045	85	43.70	1.6
ResNet18-FashionMNIST	94.88	92.43	0.045	86	43.68	1.6
ResNet18-CIFAR10	93.02	88.89	0.045	73	43.705	1.6
ResNet18-CIFAR100	78.32	73.96	0.045	60	43.885	1.7
VGG16-MNIST	97.31	95.55	0.05	86	57.575	1.9
VGG16-FashionMNIST	94.01	91.91	0.05	86	57.574	1.9
VGG16-CIFAR10	92.29	87.76	0.05	78	59.0	2.0
VGG16-CIFAR100	70.48	65.32	0.05	76	59.759	2.1

In short, although the recognition accuracy of the binary branch has gaps with the main branch, LCRS leverages the collaboration of the main branch located at the edge server to supply the shortage of the binary branch. Simultaneously, a lightweight binary branch provides a crucial foundation for executing deep neural networks on the mobile web browser in real-time.

We present the training performance of the binary branch of various networks and datasets in Figure 5. Almost networks perform rapid convergence and converge in early time. LeNet performs advantages on MNIST and FashionMNIST while deeper networks have advantages on CIFAR10 and CIFAR100. Generally, the training performance of the binary branch has a similar trend to a full precision branch.

B. Latency performance

We discuss the average latency of LCRS on various networks over datasets in Figure 6. The input sample can be processed and exit from the binary branch directly or collaborate with the edge server for more precise inference. The results reveal that average latency is almost stable because exiting probability is fixed for each binary branch in spite of the increase of samples. However, as a result of low accuracy of binary branches, partial samples have to transfer intermediate results to the edge server to acquire satisfactory inference. Also, communication costs lead to fluctuations in

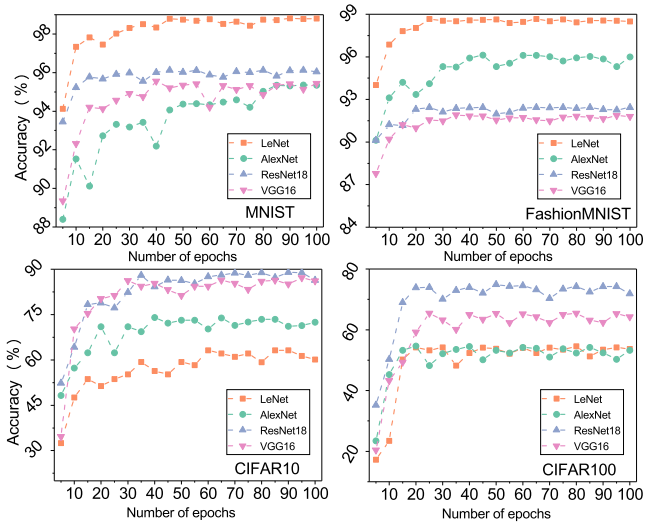

Figure 5. Training performance of the binary branch.

average latency. To sum up, this collaborative mechanism ensures real Web AR applications with low latency and high precision.

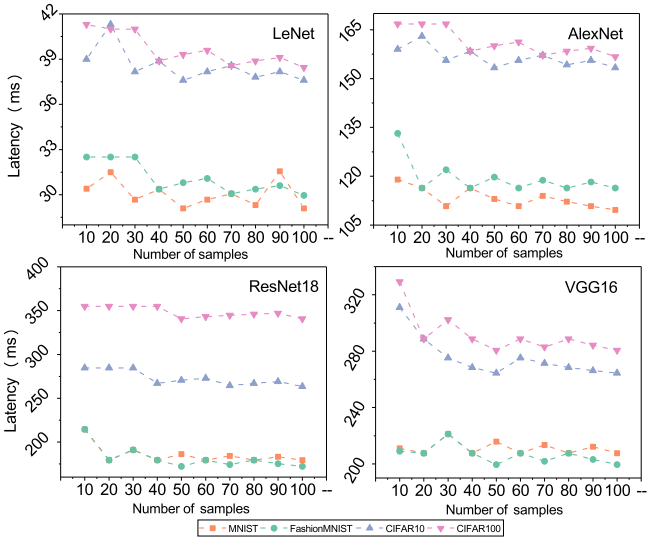

Figure 6. Average latency performance.

Table II discusses the latency performance of LCRS against Neurosurgeon, Edgent and Mobile-only using the average latency of 100 random samples. The network setting is in 4G with a downlink of 10Mb/s and an uplink of 3Mb/s. The binary branch accelerates the execution and decreases the whole inference latency. Especially, for deeper neural networks (e.g., ResNet18 and VGG16), because the computing resource of the mobile web browser is limited, traditional full precision networks are invalid executing on the mobile web browser. Although Neurosurgeon and Edgent load and execute partial layers of deep neural networks for fast inference, they also become invalid facing deeper networks because the model size is still too large to load and execute efficiently.

Table II
AVERAGE LATENCY EXECUTING ON MOBILE WEB BROWSER

-	LCRS	Neurosurgeon	Edgent	Mobile-only
LeNet	37	110	204	109
AlexNet	153	5256	4617	9313
ResNet18	261	2820	2613	5882
VGG16	264	3421	3231	8205

[a]The unit of measurement is ms.

In Table III, we discuss communication costs including loading model, transferring intermediate results and initial task by evaluating the average cost of 100 random samples. It is obvious that the communication cost of the full precision deep neural network is too high to accept in the current environment. Especially, mobile-only, Neurosurgeon and Edgent become weak which consumes much time on loading model while LCRS is stronger to handle with deeper networks. One the one hand, LCRS reduces loading model latency and accelerates inference on the mobile web browser by binarizing the convolutional layers. On the other hand, the collaboration between the mobile web browser and the edge server ensures the inference accuracy which only transfers intermediate convolutional results rather initial task.

Table III
AVERAGE COMMUNICATION COSTS EXECUTING ON MOBILE WEB BROWSER

-	LCRS	Neurosurgeon	Edgent	Mobile-only
LeNet	19	72	56	170
AlexNet	340	512	492	9104
ResNet18	188	297	287	4406
VGG16	234	365	324	5832

[a]The unit of measurement is ms.

We compare the model size of various networks of CIFAR10 executing on the mobile web browser over the aforementioned approaches in Figure 7. For Neurosurgeon and Edgent, we leverage the same partition points described in the literature. The bar charts clearly show that full precision deep networks are not appropriate to execute on the mobile web browser even for partition-offloading methods, model size and inference computation are too heavy to execute. Thus, LCRS's binary branch aims to alleviate this situation with a lightweight model and fast inference on the mobile web browser. Our evaluations indicate that we can complete the effective recognition in one second. Especially

in deeper networks such as AlexNet, ResNet18 and VGG16, LCRS performs great advantages on the overall latency.

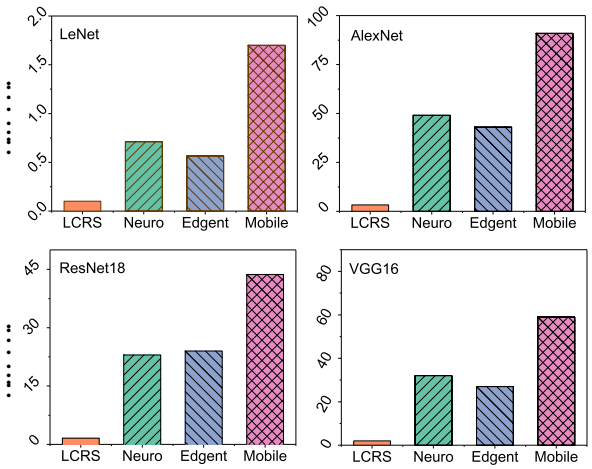

Figure 7. Model size of various approaches on CIFAR10.

C. Applications of Web AR

In Figure 8, IBM X3640M4 equipped with E5-2640 at 2.9 GHz and RAM of 16 GB is deployed at the edge server. We further use HUAWEI Mate 9 equipped with CPU at 2.4GHz and run the Android system with Firefox browser. We leverage LCRS to recognize the real cases of China Mobile and FenJiu in Figure 9.

Figure 8. The topology of Web AR applications.

The main process of two cases consists of scanning, recognition and rendering. We collect a batch of logos of China Mobile and use data augmentation techniques such as rotation, translation, zoom, flips and colour perturbation to expand the dataset. The similar process is used in FenJiu case of the training phase of LCRS. Users can scan the logo of China Mobile and the wine bottles of FeiJiu to experience the Web AR applications. The results show that the average latency of the whole process can limit in one second, which recognition reduces most of the latency against the aforementioned approaches.

We discuss detail latency performance of recognition in China Mobile case using ResNet18 in Figure 10. LCRS-B and LCRS-M denote the input sample exiting from the binary branch and the main branch respectively. The results reveal that leveraging binary convolutional network for the mobile web browser has larger promotions than existing DNN executing frameworks.

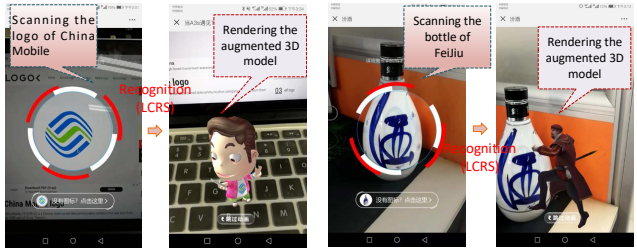

Figure 9. Web AR applications.

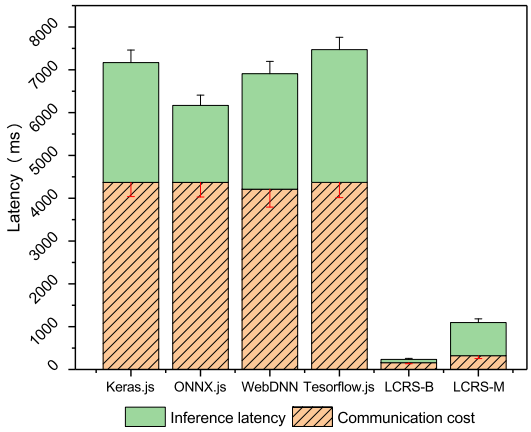

Figure 10. Latency performance of recognition in Web AR applications

VI. Conclusions and Future Work

In this work, we proposed LCRS, a lightweight collaborative recognition system with binary convolutional neural network for Web AR. Towards memory-saving and low-latency, LCRS introduces binary convolutional neural network into a typical deep neural network for reducing the memory usage and accelerating inference. We also implement an inference library for running LCRS on the mobile web browser. Moreover, we leverage the collaboration of the main branch located at the edge server to supply the shortage of the binary branch. Because loading latency is ignored in existing approaches, the results show that LCRS performs advantages in terms of memory usage and inference latency in practical Web AR applications. These results give us insightful motivation to expend LCRS on more complex networks and images. In future research, we need to validate the effectiveness of LCRS on recognition of more categories. Furthermore, we may do more simulation in different system environments for more insightful knowledge.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61671081, in part by the Funds for International Cooperation and Exchange of NSFC under Grant 61720106007, in part by the 111 Project under Grant B18008, in part by the

Beijing Natural Science Foundation under Grant 4172042, in part by the Fundamental Research Funds for the Central Universities under Grant 2018XKJC01, and in part by the BUPT Excellent Ph.D. Students Foundation under Grant CX2019135. The work of L. Liu was supported in part by the National Science Foundation under Grant NSF 1547102 and Grant SaTC 1564097 and in part by the IBM Faculty Award.

REFERENCES

[1] X. Qiao, P. Ren, S. Dustdar, et al. "Web AR: A Promising Future for Mobile Augmented Reality–State of the Art, Challenges, and Insights," Proceedings of the IEEE, Vol. 107, pp. 651-666, April 2019.

[2] C. Inmaculada, C. Sergio, C. Pablo, et al, "FI-AR learning: a web-based platform for augmented reality educational content," Multimedia Tools and Applications, pp. 1-26, July 2018.

[3] D. Lowe, "Distinctive image features from scale-invariant keypoints," International journal of computer vision, Vol. 60, pp. 91-110, November 2004.

[4] H. Bay, T. Tuytelaars, G. Van, "Surf: Speeded up robust features," European conference on computer vision. Springer, Berlin, pp. 404-417, May 2006.

[5] J. Morel, G. Yu, "ASIFT: A new framework for fully affine invariant image comparison," SIAM journal on imaging sciences, Vol. 2, pp. 438-469, January 2009.

[6] P. Ren, X. Qiao, J. Chen, et al, "Mobile Edge Computing–a Booster for the Practical Provisioning Approach of Web-Based Augmented Reality," 2018 IEEE/ACM Symposium on Edge Computing (SEC). IEEE, pp. 349-350, October 2018.

[7] X. Qiao, P. Ren, S. Dustdar, et al, "A New Era for Web AR with Mobile Edge Computing," IEEE Internet Computing, Vol. 22, pp. 46-55, August, 2018.

[8] Keras.js, https://github.com/transcranial/keras-js, 2016.

[9] Tensorflow.js, https://js.tensorflow.org/, 2018.

[10] Y. Kang, J. Hauswald, C. Gao, et al, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," ACM SIGPLAN Notices, Vol. 52, pp. 615-629, April 2017.

[11] H. Masatoshi, K. Yuichiro, U. Yoshitaka, and H. Tatsuya, "Webdnn: Fastest dnn execution framework on web browser," Proceedings of the 2017 ACM on Multimedia Conference, ACM, pp. 1213–1216, October 2017.

[12] ONNX.js, https://github.com/Microsoft/onnxjs, 2018.

[13] M. Chris, "Webgl specification,". Khronos WebGL Working Group, 2011.

[14] A. Haas, A. Rossberg, D. Schuff, et al, "Bringing the web up to speed with WebAssembly," ACM SIGPLAN Notices. ACM, Vol. 52, pp. 185-200, June 2017.

[15] E. Li, Z. Zhou, X. Chen. "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," Proceedings of the 2018 Workshop on Mobile Edge Communications. ACM, pp. 31-36, June 2018.

[16] S. Han, H. Shen, M. Philipose, et al, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services. ACM, pp. 123-136, June 2016.

[17] J. Mao, X. Chen, K. Nixon, et al, "Modnn: Local distributed mobile computing system for deep neural network," 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, pp. 1396-1401, March 2017.

[18] A. Eshratifar, M. Abrishami, M. Pedram, "JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services," arXiv preprint arXiv:1801.08618, January 2018.

[19] D. Soudry, I. Hubara, R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," Advances in Neural Information Processing Systems, Vol. 2, pp. 963-971, September 2014.

[20] S. Esser, R. Appuswamy, P. Merolla, et al, "Backpropagation for energy-efficient neuromorphic computing," Advances in Neural Information Processing Systems, pp. 1117-1125, December 2015.

[21] M. Courbariaux, Y. Bengio, J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," Advances in neural information processing systems, pp. 3123-3131, November 2015.

[22] M. Rastegari, V. Ordonez, J. Redmon, et al, "Xnor-net: Imagenet classification using binary convolutional neural networks," European Conference on Computer Vision, Springer, Cham, pp. 525-542, October 2016.

[23] S. Teerapittayanon, B. McDanel, H. Kung, "Distributed deep neural networks over the cloud, the edge server and end devices," International Conference on Distributed Computing Systems, IEEE, pp. 328-339, September 2017.

[24] M. Courbariaux, Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," eprint arXiv:1602.02830, February, 2016.

[25] CaffejJS, https://github.com/chaosmail/caffejs, 2016.

[26] S. Teerapittayanon, B. McDanel, H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," 23rd International Conference onPattern Recognition, IEEE, pp. 2464-2469, September 2017.

[27] K. He, X. Zhang, S. Ren, et al, "Deep residual learning for image recognition," Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778, June 2016.

[28] A. Zakai, "Emscripten: an LLVM-to-JavaScript compiler," Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, ACM, pp. 301-312, October 2011.