A Dynamic Deep Neural Network Design for Efficient Workload Allocation in Edge Computing

Chi Lo, Yu-Yi Su, Chun-Yi Lee, and Shih-Chieh Chang
Dept. of Computer Science, National Tsing Hua University
No. 101, Sec. 2, Kuang-Fu Rd., Hsinchu, Taiwan 30013, R.O.C.
{chilo9212, wwball34}@gmail.com, {cylee, scchang}@cs.nthu.edu.tw

***Abstract*—Unreliable communication channels and limited computing resources at the edge end are two primary constraints of battery-powered movable devices, such as autonomous robots and unmanned aerial vehicles (UAVs). The impact is especially severe for those performing deep neural network (DNN) computations. With increasing demand for accuracy, the trend in modern DNN designs is the use of cascaded modularized layers. Implementing a deep network at the edge increases computational workloads and resource occupancy, leading to an increase in battery drain. Using a shallow network and offloading workloads to backbone servers, however, incur significant latency overheads caused by unstable communication channels. Hence, dynamic DNN design techniques for efficient workload allocation are urgently required to manage the amount of workload transmissions while achieving the required accuracy. In this paper, we explore the use of authentic operation (AO) unit and dynamic network structure to enhance DNNs. The AO unit defines a set of stochastic threshold values for different DNN output classes and determines at runtime if an input has to be transferred to backbone servers for further analysis. The dynamic network structure adjusts its depth according to channel availability. Experiments have been comprehensively performed on several well-known DNN models and datasets. Our results show that, on an average, the proposed techniques are able to reduce the amount of transmissions by up to 17% compared to previous methods under the same accuracy requirement.**

***Keywords*—Deep neural network, workload allocation, edge computing, authentic operation, dynamic network structure**

I. Introduction

Deep neural networks (DNNs) have emerged as a popular design paradigm in the area of image classification and object detection [1]–[12]. This is due to their capabilities to extract high-level and abstract features from raw data. A number of DNN architectures and training algorithms have been proposed to improve the accuracy of multilayer perceptrons (MLPs) and convolutional neural networks (CNNs) from different perspectives [3]–[5]. With increasing demand for high accuracy, there has been a trend in recent years to increase the number of layers of DNNs. Several state-of-the-art CNN architectures, such as AlexNet [2], Network In Network (NIN) [6], VGGNet [7], and GoogLeNet [8], contain from eight to dozens of hidden layers. Researchers have even further pushed the network size up to 152 layers [9], achieving an unprecedented error rate less than 5% on the famous ImageNet dataset [13]. It is believed that the deeper a network is, the higher the accuracy it delivers [7]–[9]. However, deeper neural networks usually require more computation, leading to higher workloads as well as resource occupancy compared to shallower ones. These computational workloads and resource requirements limit the scale of DNNs to be executed on energy-constrained embedded devices. Thus, an efficient method to manage the workloads of embedded systems performing DNN computations is urgently needed.

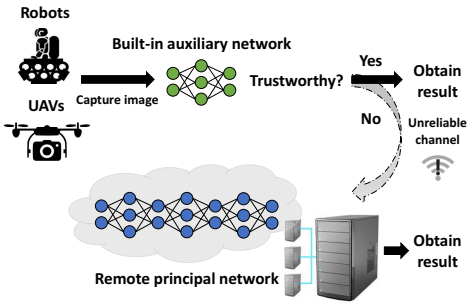


Fig. 1: Workload sharing of DNN between edge and server

The concept of *edge computing* [14]–[16] is to perform data processing at the edge end, near the source of data. Edge-end embedded devices, such as unmanned aerial vehicles (UAVs) and autonomous robots, work synergistically with powerful servers to provide performance in modern edge computing systems. Edge-end devices usually have limited computational capability and resources, thus only shallower DNNs (denoted as *auxiliary networks*) can be accommodated, as compared to the deeper ones (denoted as *principal networks*) executed at the server end. In such systems, the edge devices may suffer from unstable communication channels. When the communication channels are fully accessible, edge devices can leverage both the auxiliary networks and principal networks to achieve high DNN accuracy. When the channels are unstable, edge devices can only share fewer workloads to the server. As a result, utilizing the communication channels and efficiently allocating workloads between an auxiliary network and a principal network are of particular importance.

A promising strategy to deal with the above issues is to allocate difficult DNN workloads to the principal network at the remote server, while retaining easy ones at the edge. Whether to transfer an input from the edge to the server is determined by calculating its *confidence level*. Confidence level is used as a measure of the reliability of a prediction. The higher the confidence level is, the more trustworthy the prediction might be. Fig. 1 illustrates such a scenario. The input images captured by robots or UAVs are first analyzed

1063-6404/17 $31.00 © 2017 IEEE
DOI 10.1109/ICCD.2017.49

by their built-in auxiliary networks. If the results indicate that confidence levels are high enough, i.e., trustworthy, they are directly reported at the edge. Otherwise, they are transferred to the remote server to be further analyzed by the principal network. Since only few input data are allowed to be sent to the server due to unreliable channels, an efficient scheme to determine the confidence level of an input is necessary.

A few past researches used similar concepts for energy or performance optimization [17]–[19]. The authors of these works proposed an idea called *fast inference* or *fast existing*. The idea suggests calculating the confidence level of an image at branch points of DNNs, and comparing their values with a predefined threshold. If the confidence level is higher than the threshold, a prediction of the image could be made with fewer layers of neural networks. Otherwise, the image has to go through the entire network to make a prediction. To the best of our knowledge, none of the previous works had applied this concept to distribute workloads in the scenario of edge computing.

To distribute workloads efficiently, we propose two techniques, called *authentic operation (AO)* and *dynamic network sizing*. AO determines whether input images have to be transferred to the principal network. Unlike the previous works, AO improves the way of setting a threshold by a statistical method, rather than adopting an user-defined one. Furthermore, AO introduces a novel concept of applying different thresholds for different classes based on the difficulties. With AO, we are able to obtain fine-grained thresholds and enhance the efficiency of workload distribution. The improvement leads to higher *overall accuracy*, which is defined as the combined accuracy derived from the accuracies of both the auxiliary and principal networks.

On the other hand, dynamic network sizing adjusts the number of layers of an auxiliary network to utilize the communication channels efficiently. Two modes of auxiliary network are available in our design: *normal mode* and *workload sharing (WS) mode*. Under the normal mode, which is enabled when the communication channels are unstable, the inputs have to traverse the entire network to ensure the accuracy at the edge. On the contrary, when the channels are fully available, the auxiliary network switches to the WS mode. The inputs only go through a fraction of the auxiliary network, such that the percentage of inputs reported at the edge is reduced. This allows more workloads to be transferred to the principal network. Dynamic network sizing intelligently determines the size of the network on the basis of the availability of the communication channels.

To verify our proposed techniques, we have conducted experiments on several well-known DNN models (e.g., NIN [6], Residual Network (ResNet) [9], Wide Residual Networks (WRN) [10], etc.) and datasets (e.g., CIFAR-10 [20] and CIFAR-100 [20]). The results show that our method is able to reduce the amount of transmission by up to 17% compared to the previous methods under the same accuracy requirement. Our architecture is specifically suitable for embedded systems. Though the proposed techniques can be applied to any DNN structures, we use CNNs to illustrate our ideas.

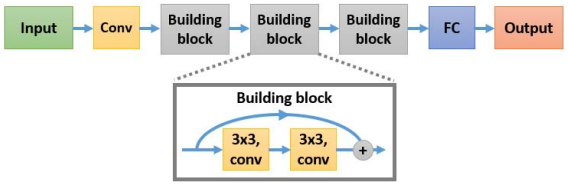


Fig. 2: Structure of an eight-layer ResNet and its building blocks

The contributions of this work include:

- **A strategy to efficiently allocate workloads of DNN computations under unreliable channels**
- **An AO unit to apply different thresholds to different DNN output classes**
- **A concept of dynamically adjusting the size of network according to channel availability**

The remainder of this paper is organized as follows. Section II introduces background material. Section III walks through the basic idea of AO and the design decisions made. Section IV discusses the dynamic network sizing scheme. Section V presents experimental results. Section VI concludes.

II. Background

A. CNN and Softmax Function

LeNet-5 [1] was one of the pioneers of CNN. In recent years, researchers have explored a number of innovative extensions, such as AlexNet [2], VGGNet [7], GoogLeNet [8], and ResNet [9]. These works pushed CNN researches to new frontiers and benefited several computer vision applications. In addition to adjusting CNN models, a great number of algorithms [4], [5] have been proposed in order to further optimize the performance of CNNs. With these contributions, CNNs have achieved unprecedented accuracy [9] due to their capabilities to extract high-level and abstract features.

A CNN generally consists of convolutional layers, pooling layers and fully-connected layers. A *Softmax function* layer is usually applied to the final layer of a CNN. It is defined as:

$$y_i = softmax(z_i) = \frac{exp(z_i)}{\sum_C exp(\bar{z})} \quad (1)$$

where C is the number of output classes, i the index to the classes, $\bar{z}$ the original outputs of all classes, z_i the original output of class i, and y_i the softmax output of class i. This softmax function takes as input a C-dimensional vector $\bar{z}$ and outputs a C-dimensional vector y of real values between 0 and 1. This function is a normalized exponential and its denominator acts as a regulator to ensure that $\sum y_i = 1$. Softmax outputs are generally interpreted as indicators of the extent of CNNs confidence for different output classes. As a result, they can serve as a measure of confidence levels.

B. Modularized Components in CNNs

The recent trend of designing CNNs is to cascade modularized components repeatedly to build the network models [3], [6], [8]–[10]. Modularized CNN structures relieve researchers from designing hyper-parameters for each layer

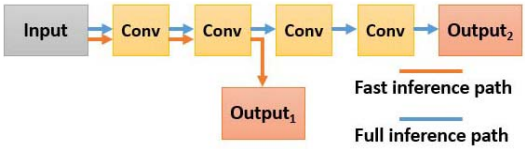


Fig. 3: Illustration of the concept of fast inference

of the network. Instead, they only need to design the hyper-parameters of modularized components. Fig. 2 shows an example of an eight-layer ResNet, which consists of three modularized components called building blocks. Each building block contains two to three convolutional layers, along with a bypass path. It is believed that modularized design styles are capable of delivering higher accuracy than traditional ones. Modularized CNN structures also allow branches to be pulled out between the components, enabling a number of interesting design innovations. In this paper, we also utilize this feature in our dynamic network sizing technique.

C. Fast Inference

The concept of fast inference is based on the assumption that the number of network layers traversed by an input is positively correlated to its classification accuracy [19]. Assume that an image dataset contains only three classes of objects: vehicle, cat, and dog. Vehicle is a class relatively easy to be classified since its features are different from those of the other classes. Fast inference utilizes this observation by predicting most inputs from vehicles class with fewer layers while predicting the rest from this class with more layers. The number of layers which an input image has to traverse is decided by comparing its confidence level with a pre-determined threshold. The concept is illustrated in Fig. 3, which contains a fast inference path (denoted in orange color) and a full inference path (denoted in blue color). If the confidence level of an input image calculated at the branch point is higher than the threshold, the prediction is made and directly reported by the fast inference path. Otherwise, the input has to go through the full inference path. We briefly introduce two relevant works [17], [18] utilizing the concept of fast inference, as they are compared with our proposed technique in Section V.

The authors in [17] proposed a technique called Conditional Deep Learning (CDL), which directly generates probabilities as the confidence levels at the output end. To determine whether the classification task is allowed to be terminated at a branch output, a user-defined threshold for the highest confidence level was set as the criterion. It is expressed as:

$$Max\_Confidence\_Level > Threshold \tag{2}$$

If the highest confidence level is greater than the threshold, it's believed as a trustworthy prediction. Unfortunately, if the confidence levels of two or more classes are close, CDL may lead to incorrect predictions. Besides, a single threshold is a coarse criterion. A detailed analysis on this issue is in Section III.

The authors from [18] proposed a structure called BranchyNet to provide more flexibility at the branch points. Similarly, a user-defined threshold is used as the criterion. It is defined as:

$$Entropy(softmax\_outputs) < Threshold \tag{3}$$

where $softmax\_outputs$ is a set which consists of softmax outputs from all of the classes. The entropy function is expressed as:

$$Entropy(\bar{X}) = \sum_{C} X_i \log(X_i) \tag{4}$$

The entropy function considers the overall distribution of the softmax outputs. The more centralized the overall distribution of the softmax outputs is, the lower the entropy it corresponds to. Since BranchyNet only considers the overall distribution, it is not capable of determining an appropriate threshold value. Similarly to CDL, the input images leave BranchyNet (the small network) if their confidence levels are less than a pre-defined entropy threshold.

Please note that both [17] and [18] focus on optimizing the fast inference structures. In this paper, we emphasize on optimizing the threshold values and allocating the workload between an auxiliary network and a principal network.

III. Authentic Operation Unit

In this section, we present a detailed design of the AO unit, an arbitration unit which significantly improves workload allocation efficiency of an auxiliary network. AO is a functional unit used to determine if the prediction from an auxiliary network is trustworthy for an input. The basic idea behind AO is to apply different threshold criteria of confidence level for different CNN output classes. An input with a confidence level higher than the threshold for a specific class is considered trustworthy, and its predicted classes is reported by the edge. An input with confidence levels lower than the thresholds, which implies insufficient capability of the auxiliary network to perform classification, is transferred to the deeper principal network. In other words, different inputs may traverse different number of CNN layers, thus incur different amount of workloads. This has two advantages. Firstly, the decision accuracy at the edge is maintained because only the inputs with high confidence levels are kept. Secondly, fine-grained workload allocation is achieved since AO adapts confidence level threshold for different classes. Though the scheme is applicable and generalizable to many scenarios, such as object detection and semantic segmentation, we illustrate its effectiveness by image classification tasks.

A. AO Decision Flow

Fig. 4 illustrates the decision flow of AO. AO is placed at the output of an auxiliary network. An input to an auxiliary network is first obtained from sensors or cameras at the edge. The input is then processed and injected into the auxiliary network to generate a vector of probabilities. To predict a class label, an $argmax$ operation is performed on the vector to retrieve the index of the maximum probability across all labels. After computation is done, AO checks if the prediction is trustworthy. If so, the auxiliary network reports the predicted result. Otherwise, it transfers the input to the server. Please

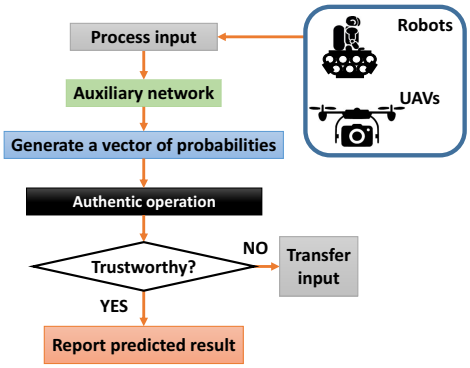
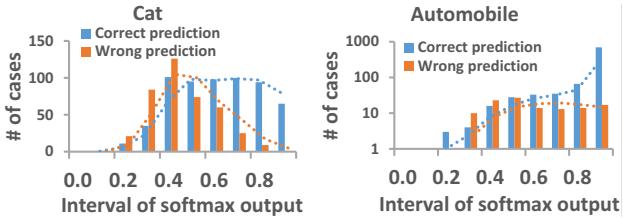


Fig. 4: Decision flow of authentic operation



(a) Histogram: Cat (b) Histogram: Automobile

Fig. 5: Histograms of max softmax outputs

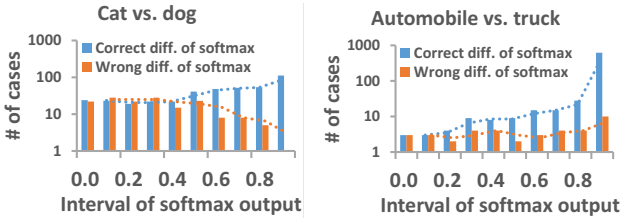
note that it is impractical to transfer intermediate feature maps directly to the server because their data size is typically large. An efficient AO not only separate difficult inputs from easy ones, but also maintains the overall accuracy of the entire system.

The key to designing an efficient AO lies in its decision criterion to determine if a prediction is trustworthy. A strict criterion leads to over-offloading to the principal network, since only a small portion of predictions are reported at the edge. On the contrary, a loose criterion results in degradation of accuracy. Thus, for an effective trade-off, fine-grained AO criteria for different output classes are necessary.

B. Fine-Grained AO Criteria

The aim of AO criteria is to determine the best thresholds of confidence level for different output classes. A good AO criteria would efficiently use edge resources to achieve high accuracy, but leave difficult inputs to the server. The inputs to an auxiliary network do not always result in high confidence levels for all classes. This means that for a class with a higher probability of confidence levels, a higher threshold can be set to maintain the accuracy of the auxiliary network. A class with a higher variance in confidence levels, however, a lower threshold becomes necessary to save offloading workloads. Our AO criteria customizes thresholds for different classes to adapt to their softmax distributions. Thus, most inputs with high confidence levels are directly reported at the edge. This allows the accuracy of the auxiliary network to approach that of the principal network, yet save significant amount of workloads to be transferred to the server.

Fig. 5 shows two example histograms of softmax probability distributions for two classes cat and automobile in CIFAR-10 [20], a dataset consists of 60,000 32×32 color images in 10 classes, with 5,000 training and 1,000 test images per class. The blue and orange bins represent the number of cases of correct (true positive) and incorrect (false positive) predictions, respectively. The horizontal axis is the internal of softmax outputs, ranging from 0.0 to 1.0 with an interval length equal to 0.1. The height of each bin indicates the number of images falling into that interval. From Figs. 5(a) and 5(b), it is observed that the difference in height between the blue and orange bins increases as the value of softmax output increases. For the automobile case in Fig. 5(b), this difference demonstrates a near-exponential growth as the value of the softmax output increases, suggesting that a higher threshold is appropriate. On the other hand, the cat case in Fig. 5(a) does not show such a trend, indicating that the auxiliary network is less capable of classifying this class of images. As a result, to save offloading workloads, a lower threshold is required for the cat class.



(a) Histogram: Cat vs. dog (b) Histogram: Automobile vs. truck

Fig. 6: Histograms of differences of the first and second highest softmax outputs

Based on the above observations, we propose fine-grained AO criteria with pair-wise thresholds for different output classes. Suppose that the first and second highest softmax outputs from a given input are S_a and S_b, where a and b are the classes they belong to respectively. Instead of computing a uniform threshold for all output classes, we first define a pair-wise confidence level as:

$$conf\_level_{(a,b)} = S_a - S_b \tag{5}$$

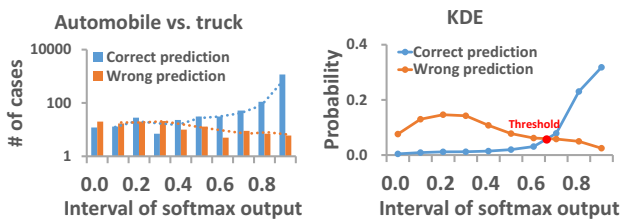
where $conf\_level_{(a,b)}$ is the difference of S_a and S_b. A pair-wise threshold $T_{(a,b)}$ is defined as the threshold for $conf\_level_{(a,b)}$, such that the AO criteria is satisfied whenever the following condition is met:

$$conf\_level_{(a,b)} >= T_{(a,b)} \tag{6}$$

The idea behind Eq. (5) is to avoid the condition when more than one softmax outputs meet the AO criteria. This ensures that only one output class with sufficient confidence level is reported at the edge. On the other hand, the different distributions of Figs. 6(a) and 6(b) lead to the idea of pair-wise thresholds. Inputs that do not satisfy Eq. (6) are forwarded to the server end. Note that the thresholds for different pairs of classes are set separately.

C. AO Threshold Selection by Kernel Density Estimation

Based on the above observations, a pair-wise threshold $T_{(a,b)}$ has to be properly selected for each pair of classes. The aim of threshold selection is to find a value such that



(a) Histogram: Automobile vs. truck (b) PDF: Automobile vs. truck

Fig. 7: AO threshold selection by KDE

the probability of making a correct prediction is higher than that of making an incorrect one. It is impractical to manually set thresholds for different pair of classes, especially when the number of classes is large. Instead of applying artificial thresholds as the previous works do [17], [18], we set these pair-wise thresholds using kernel density estimation (KDE).

KDE is used to estimate the probability density function (PDF) of random variables. We recommend the interested readers [21] for the details of KDE. Fig. 7 shows an example distribution of $conf\_level_{(automobile,truck)}$ for both correct and incorrect predictions, and the PDF of them generated by KDE. The kernel function is gaussian, and the bandwidth is set to 0.05. We can see from Fig. 7(b) that the probability of making a correct prediction grows with $conf\_level_{(automobile,truck)}$. The probability of incorrect predictions shows the opposite trend. To filter out most of the incorrect predictions, we set the intersection of PDFs of the correct prediction and incorrect prediction as the pair-wise threshold $T_{(automobile,truck)}$.

Sufficient amount of data are required to perform KDE analysis. It is difficult to set pairwise thresholds when there are too many classes in a dataset, e.g., CIFAR-100 and ImageNet. In such cases, a single threshold is set for each class, rather than in a pair-wise fashion. Eq. (6) is then rewritten as:

$$conf\_level_{(a,b)} \geq T_{(a)} \tag{7}$$

where a represents the class with the highest softmax output. The method of calculating confidence levels remains the same. However, the confidence levels are compared with the threshold indexed by the class with the highest softmax output.

IV. Dynamic Network Sizing

Dynamic network sizing is a technique used to dynamically manage the number of layers in an auxiliary network to utilize the unstable communication channels. This technique varies the depth of the auxiliary network to adjust the amount of workloads to be transferred to the principal network while maintaining the overall accuracy. When the communication channels are fully accessible, the auxiliary network becomes shallower (i.e. less accurate) to share more workloads to the principal network (i.e. more accurate and less computation energy). When the channels are unstable, however, the inputs have to go through the full length of the auxiliary network and share as less workloads as possible. This technique works with AO to regulate the amount of workloads transferred to the server, and ensure that only the difficult ones are sent.

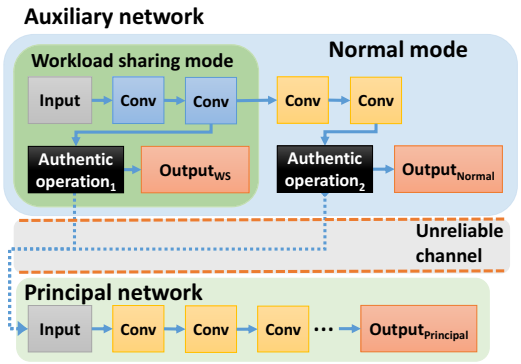


Fig. 8: Illustration of dynamic network sizing and the structure of auxiliary network

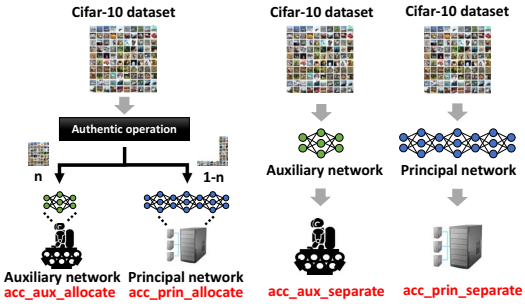


Fig. 9: Illustrations of four kinds of accuracies

A. Normal Mode and Workload Sharing Mode Structure

Two modes of auxiliary network size are available in our design: normal mode and WS mode. Fig. 8 illustrates the concept of our dynamic network sizing technique and the two modes of auxiliary network. A normal mode network corresponds to the entire auxiliary network, whereas a WS mode network corresponds to only a part of it. In the first scenario, inputs traverse more CNN layers, leading to more computational workloads and higher accuracy. In the second scenario, inputs pass through less CNN layers, leading to less computational workloads and lower accuracy. An additional AO unit and output layer are inserted in the middle of the network. When the communication channels are fully accessible, WS mode is enabled. In case of unstable connection between the edge and the server, the auxiliary network switches itself to the normal mode to maintain accuracy. The use of WS mode enables the edge to offload more workloads to the server.

B. Network Sizing Methodology

Based on the above auxiliary network structure, we present a sizing methodology to determine the best length of WS mode network. A good WS network size would efficiently use edge resources to achieve high accuracy but leave difficult inputs to the server. Table I shows the impact of network sizing on computational workloads and accuracy for three types of CNN architectures trained on the CIFAR-10 and CIFAR-100 dataset. The second column corresponds to the number of network layers, where the baselines are those used in the normal mode. The baselines vary for different network types because of their difference in architecture. The third and fourth columns

TABLE I: Impact of network sizing on workloads and accuracy

Network type	# of WS mode layers	# of norm fp-ops	Accuracy
CIFAR-10 ResNet	10	47%	73%
	14	63%	77%
	22	75%	83%
	baseline = 32	100%	86%
CIFAR-10 NIN	3	26%	42%
	6	86%	78%
	baseline = 9	100%	79%
CIFAR-100 WRN	10	48%	55%
	14	74%	60%
	baseline = 18	100%	63%

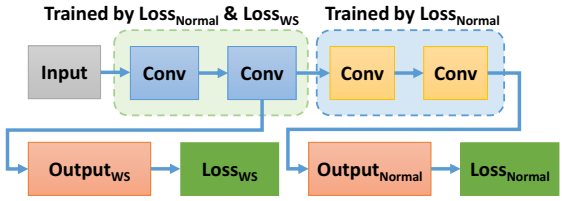


Fig. 10: Illustration of training process with multiple loss layers

report the number of floating point operations (i.e. fp-op) and network accuracy, respectively. The number of floating point operations is normalized with respect to those of the baseline, and serves as a measure of workloads. It can be seen that the less the number of network layers is, the less the workloads it generates, and the lower accuracy it delivers. This means that trading off accuracy for aggressive workload sharing can be achieved if a shallow network is adopted for the WS mode.

To illustrate the impact of network sizing, we define the accuracy of the auxiliary network and principal network as $acc_{aux\_separate}$ and $acc_{prin\_separate}$, respectively. Both $acc_{aux\_separate}$ and $acc_{prin\_separate}$ are standalone accuracy, and are measured separately from the entire test set of the dataset. We also define their accuracy after AO as $acc_{aux\_allocate}$ and $acc_{prin\_allocate}$, accounting for the effect of workload allocation done by AO. These four parameters are illustrated in Fig. 9. The overall accuracy of the entire system is calculated as:

$$Acc_{overall} = n*acc_{aux\_allocate}+(1-n)*acc_{prin\_allocate} \quad (8)$$

where n is the ratio of the number of inputs remaining at the edge to total number of inputs. The value of n depends on WS mode network size, AO criteria, the target $acc_{allocate}$, and $acc_{overall}$. Given the AO criteria defined in Section III, an appropriate WS mode network size has to be selected with care to maintain a fixed $acc_{overall}$ and a desirable $acc_{allocate}$.

C. Auxiliary Network Training Procedure

Training an auxiliary network means iteratively minimizing prediction errors for both the normal and WS modes. Common methods of training with a single loss function at the end of the network are not sufficient for a network with multiple output branches. Training an auxiliary network with two output branches, for example, can be achieved by inserting an additional loss layer at the branch point of the WS mode network, as illustrated in Fig. 10. Thus, optimization of the auxiliary network is decomposed into two parts. Both of the two loss terms contribute gradients in the back-propagation process. The optimization goal of the training procedure is to minimize the losses from the normal mode (L_{Normal}) and the WS mode (L_{WS}) concurrently, expressed as:

$$L_{Total} = \alpha * Loss_{Normal} + (1-\alpha) * Loss_{WS} \quad (9)$$

where α is a constant $\in$ [0, 1]. Cross-entropy function is used to calculate $Loss_{Normal}$ and $Loss_{WS}$, and is expressed as:

$$Loss(y,\bar{y}) = CrossEntropy(y,\bar{y}) = \frac{-1}{C}\sum_{C} \bar{y}_i \log(y_i) \quad (10)$$

where C is the number of classes, y the set of softmax outputs, $\bar{y}$ the set of ground truth labels, and $\bar{y}_i$ and y_i correspond to the softmax output and ground truth label belonging to class i. The pseudocode of our training procedure is summarized in Algorithm 1:

Algorithm 1 :Auxiliary Network Training Procedure

```
η = learning_rate
for each Layer ∈ Auxiliary Network do
    W = Layer.weights
    if Layer ∈ Normal_mode then
        W_new = W − η∇Loss_Normal(W)
    else if Layer ∈ WS_mode then
        W_new = W − η∇L_Total(W)
    end if
end for
```

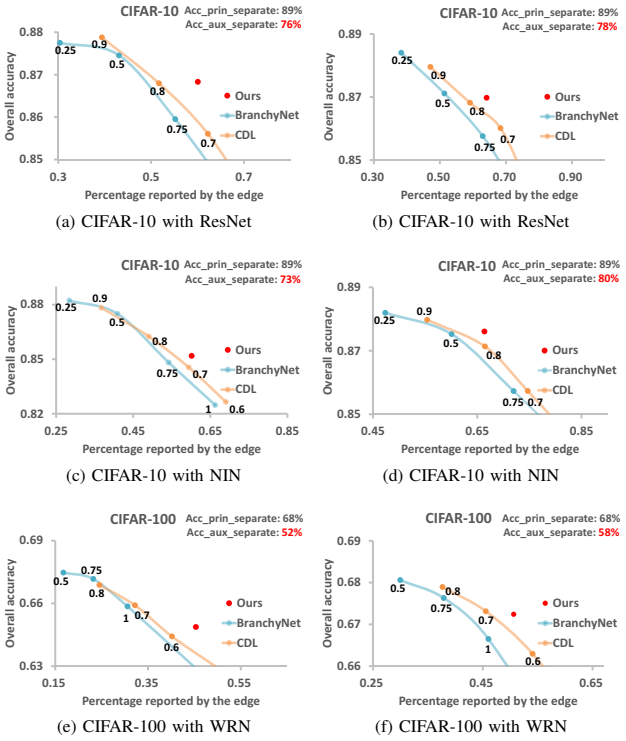
Applying two loss terms enhances the learning process, and mitigates the gradient vanishing problem usually encountered when training deep DNNs [9]. In this work, we set α to 0.5. However, other values can be set for different WS mode network lengths and network types.

V. Experimental Results

In this section, we present experimental results. The comparison of effectiveness of AO on various architectures of auxiliary networks is explored first. Then, the comparison of accuracy and shared workloads of those networks under the same overall accuracy requirement is presented. Finally, the effectiveness of dynamic network sizing is demonstrated.

A. Experimental Setup

We present our results for a number of modularized network models including NIN [6], ResNet [9], and WRN [10] on image classification datasets CIFAR-10 [20] and CIFAR-100 [20]. There are 50,000 training images and 10,000 test images with 10 and 100 classes in CIFAR-10 and CIFAR-100, respectively. Each image contains 32×32 RGB pixels. We evaluate our techniques with NIN and ResNet on CIFAR- 10, and WRN on CIFAR-100. We used Caffe [22], a framework used to process state-of-the-art deep learning algorithms, as our experiment platform. The experiments were performed on NVIDIA GeForce GTX 1070 and 1080 GPUs with 8GB of DDR5 memory, and an Intel i7-6700 CPU with 64GB of DDR4 memory. Please note that in our experiments, we use floating point operations (fp-ops) as the performance metric



(a) CIFAR-10 with ResNet (b) CIFAR-10 with ResNet

(c) CIFAR-10 with NIN (d) CIFAR-10 with NIN

(e) CIFAR-100 with WRN (f) CIFAR-100 with WRN

Fig. 11: Overall accuracy vs. percentage of inputs remaining at edge

for various networks. Therefore, the results are the same for embedded systems.

B. Comparison of Different Threshold Criteria on Various Network Architectures

We compare three different threshold criteria from CDL [17], BranchyNet [18], and ours on the three different auxiliary network architectures (NIN, ResNet, and WRN). The results are plotted in Fig. 11. The horizontal axis is the percentage of classification results directly reported at the edge. The vertical axis represents the overall accuracy defined in Eq. (8). For each network structure, we present results with two different values of $acc_{aux\_separate}$. For example, Figs. 11(a) and 11(b) show the results for $acc_{aux\_separate}$ equal to 76% and 78%, respectively. These values of $acc_{aux\_separate}$ are resulted from different depths of auxiliary network size. In Fig. 11, both CDL and BranchyNet employ a fixed threshold for all classes. Their threshold can be configured to different values. We plot their results together with ours, with the values of their suggested thresholds labeled on the figures. It is observed that under the same overall accuracy requirements, our fine-grained AO criteria is able to report more classification results than CDL and BranchyNet. When the percentage of classification results at the edge is fixed, our method is able to achieve a higher overall accuracy than CDL and BranchyNet.

C. Comparison of Workload Sharing Efficiency

Table II presents a detailed comparison of workload sharing efficiency for different network types and methodologies (CDL, BranchyNet, and AO). The separate auxiliary accuracies ($acc_{aux\_separate}$) and principal accuracies ($acc_{aux\_separate}$) are directly measured from the entire test data in the datasets. The allocate auxiliary accuracies ($acc_{aux\_allocate}$) are measured from the test data reported by the edge. The allocate principal accuracies ($acc_{prin\_allocate}$), on the contrary, are measured from the test data the sent to the server. We adopt the overall accuracy achieved by our technique as a reference target, and compare the percentage of reported images by the edge with those of CDL and BranchyNet. The values from CDL and BranchyNet are obtained by interpolation or extrapolation if the experimental results corresponding to the overall accuracy are not available. We define gain as the ratio of the percentage of our methods to those of the others. It can be seen that our AO outperforms the other techniques, and delivers an average gain of 13% and 21% compared to CDL and BrancyNet, respectively.

D. Validation of Dynamic Network Sizing

Table III shows a validation of dynamic network sizing for different network types. Similar to Table II, we show the separate auxiliary accuracies acc_{aux_WS} and acc_{aux_Normal} for the WS mode and normal mode auxiliary networks. The numbers of floating point operations are normalized to that of the normal mode network. The overall accuracy achieved by the normal mode network is taken as a reference target. It is observed that WS mode networks are able to reduce the amount of workloads (i.e. fp-ops), as well as transfer more input images to the server. This is reflected by the drop of percentage of the reported images at the edge in the fourth column. The dynamic sizing technique is able to reduce up to 39% of the floating point operations for the CIFAR-100 case (with WRN), and transfers 22% more images to the server when the communication channels are fully accessible. The results demonstrate effectiveness of our dynamic network sizing technique.

VI. Conclusion

We proposed a workload management method to allocate workloads transferred from an auxiliary network and to a principal network. The method consists of two techniques, AO and dynamic network sizing. By utilizing AO, we are able to achieve a high overall accuracy with a large portion of input images remaining at the edge end. In addition, dynamic network sizing enables an auxiliary network to maintain workload sharing efficiency even under the unstable communication channel. We have evaluated the proposed method on several well-known CNN architectures with the famous CIFAR-10 and CIFAR-100 datasets. On an average, the proposed AO technique reduces 17% of the transmissions under the same overall accuracy requirement compared to the previous works. With dynamic network sizing, an auxiliary network is able to reduce up to 39% of its floating point operations by transferring additional 22% of its input images to the server end. By applying AO and dynamic network sizing together, we are able to distribute workload efficiently while maintaining

TABLE II: Comparison of workload sharing efficiency for different network types and methodologies

Network type	Separate		Method	% of inputs reported at the edge	Gain	Allocate		Overall accuracy
	Auxiliary accuracy	Principal accuracy				Auxiliary accuracy	Principal accuracy	
CIFAR-10 ResNet	76.08%	89%	CDL [17]	51.30%	17.10%	93%	80%	86.83%
			BranchyNet [18]	48.12%	24.84%	93%	81%	
			Ours	**60.07%**	-	**90%**	**81%**	
	78.25%	89%	CDL [17]	57.44%	11.68%	94%	78%	86.98%
			BranchyNet [18]	52.53%	22.11%	95%	78%	
			Ours	**64.15%**		**92%**	**79%**	
CIFAR-10 NIN	73.03%	89%	CDL [17]	55.52%	8.35%	90%	79%	85.19%
			BranchyNet [18]	52.37%	14.87%	91%	79%	
			Ours	**60.16%**	-	**88%**	**81%**	
	80.17%	89%	CDL [17]	60.25%	10.20%	95%	77%	87.61%
			BranchyNet [18]	58.49%	13.50%	95%	77%	
			Ours	**66.39%**	-	**92%**	**78%**	
CIFAR-100 WRN	51.52%	68%	CDL [17]	37.75%	20.06%	80%	56%	64.87%
			BranchyNet [18]	35.47%	27.79%	80%	57%	
			Ours	**45.32%**	-	**75%**	**56%**	
	58.23%	68%	CDL [17]	46.15%	9.72%	84%	53%	67.24%
			BranchyNet [18]	41.12%	23.15%	86%	54%	
			Ours	**50.64%**	-	**81%**	**53%**	

TABLE III: Validation of dynamic network sizing for different network types

Network type	# of WS mode layers	# of norm fp-ops	% of inputs reported at the edge	Allocate		Overall accuracy
				Auxiliary accuracy	Principal accuracy	
CIFAR-10	WS: 16	63%	72%	91%	79%	88%
ResNet	Normal: 32	100%	81%	94%	67%	89%
CIFAR-10	WS: 6	86%	64%	91%	79%	87%
NIN	Normal: 9	100%	66%	92%	78%	87%
CIFAR-100	WS: 12	61%	56%	77%	53%	66%
WRN	Normal 18	100%	71%	73%	49%	66%

performance in edge computing. We leave practical implementations and runtime adjustable thresholds as our future directions.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Information Processing Systems (NIPS)*, pp. 1097-1105, Dec. 2012.

[3] F. N. Iandola *et al.*, "Squeezenet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size," *arXiv:1602.07360*, Mar. 2016.

[4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Machine Learning (ICML)*, pp. 448-456, Feb. 2015.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958, Jun. 2014.

[6] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv:1312.4400*, Dec. 2013.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, Sep. 2014.

[8] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 1-9, Jun. 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[10] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv:1605.07146*, May 2016.

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 580-587, Jun. 2014.

[12] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 3431-3440, Jun. 2015.

[13] J. Deng *et al.*, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 248-255, Jun. 2009.

[14] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *J. IEEE Internet of Things*, vol. 3, no. 6, pp. 1171-1181, Dec. 2016.

[15] P. G. Lopez *et al.*, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37-42, Oct. 2015.

[16] V. Mushunuri, A. Kattepur, H. K. Rath, and A. Simha, "Resource optimization in fog enabled IoT deployments," in *Proc. IEEE Int. Conf. Fog and Mobile Edge Computing (FMEC)*, pp. 6-13, May 2017.

[17] P. Panda, A. Sengupta, and K. Roy, "Energy-efficient and improved image recognition with conditional deep learning," *J. Emerging Technologies in Computing Systems*, vol. 13, no. 3, pp. 33:1-33:21, Feb. 2017.

[18] S. Teerapittayanon, B. McDanel, and H. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. IEEE Int. Conf. Pattern Recognition (ICPR)*, pp. 2464-2469, Dec. 2016.

[19] S. Venkataramani, A. Raghunathan, J. Liu, and M. Shoaib, "Scalable-effort classifiers for energy-efficient machine learning," in *Proc. Design Automation Conf.*, pp. 67:1-67:6, Jun. 2015.

[20] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Master's thesis*, Department of Computer Science, University of Toronto, Apr. 2009.

[21] E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics*, vol. 33, pp. 1065-1076, Sep. 1962.

[22] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. ACM Int. Conf. Multimedia*, pp. 675-678, Nov. 2014.