Complexity-aware Adaptive Training and Inference for Edge-Cloud Distributed AI Systems

Yinghan Long, Indranil Chakraborty, Gopalakrishnan Srinivasan, Kaushik Roy
School of Electrical and Computer Engineering, Purdue University
long273@purdue.edu, ichakra@purdue.edu, srinivg@purdue.edu, kaushik@purdue.edu

2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS) | 978-1-6654-4513-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICDCS51616.2021.00061

***Abstract*—The ubiquitous use of IoT and machine learning applications is creating large amounts of data that require accurate and real-time processing. Although edge-based smart data processing can be enabled by deploying pretrained models, the energy and memory constraints of edge devices necessitate distributed deep learning between the edge and the cloud for complex data. In this paper, we propose a distributed system to exploit both the edge and the cloud for training and inference. We propose a new architecture, MEANet, with a *main block*, an *extension block*, and an *adaptive block* for the edge. The inference process can terminate at either the main block, the extension block, or the cloud. MEANet is trained to categorize inputs into easy/hard/complex classes. The main block identifies instances of easy/hard classes and classifies easy classes with high confidence. Only data with high probabilities of belonging to hard classes would be sent to the extension block for prediction. Further, only if the neural network at the edge shows low confidence in the prediction, the instance is considered complex and sent to the cloud for further processing. The training technique lends to the majority of inference on edge devices while going to the cloud only for a small set of complex jobs. The performance of the proposed system is evaluated via extensive experiments using modified models of ResNets and MobileNetV2 on CIFAR-100 and ImageNet datasets. The results show that the proposed distributed model has improved accuracy and lower energy consumption compared to standard models, indicating its capacity to adapt.**

***Index Terms*—Deep Learning; Distributed Systems; Edge Computing**

I. Introduction

The availability of a large amount of data at the edge and the development of Artificial Intelligence (AI) has significantly augmented the paradigm of Internet of Things (IoT). Deep Learning (DL) and deep neural networks (DNN) have achieved remarkable performance in various applications such as computer vision, including object detection, face and visual scene recognition [1]–[3]. The growth of machine learning applications has, on one hand, positively impacted human lives, but on the other hand, has led to the need to smartly manage the humongous amount of heterogeneous data [3], [4]. Processing this data (which is available at the edge) entirely in the cloud raises concerns about latency, energy, and data privacy. Despite its success, DNNs are computationally expensive, making them unsuitable for resource-constrained edge devices. This necessitates intelligent collaborative decision strategies between the resource constrained edge devices and the cloud.

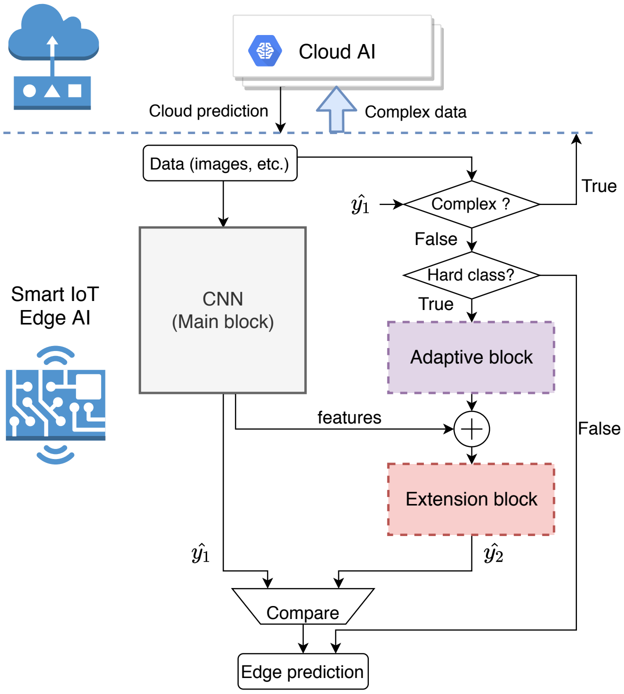

Fig. 1: Overview. The distributed system consists of a DNN on the cloud and a MEANet at the edge.

A distributed AI system with edge-cloud collaboration primarily relies on two pillars: 1) Energy-efficient computing at the edge, 2) Intelligent edge-cloud distribution. Distributed computing consisting of the edge and the cloud has inherent advantages, such as providing system flexibility and scalability, and supporting coordinated central and local decisions [5]. Edge computing and cloud computing can collaborate by allowing the edge to activate the cloud as needed. There are two ways of edge-cloud collaboration by conditionally sending the raw data (e.g. images) or processed features to the cloud. The first way allows the edge network and cloud network to be relatively independent. The second approach would partition a deep neural network across the edge and the cloud. The edge network can potentially be simple and suitable for classifying simple data with high accuracy. Various optimization techniques have been proposed to alleviate the computational cost of DNNs for edge computing, such as quantization and branching [6], [7]. More complex data can be conditionally sent to the cloud based on the confidence derived

The research was funded in part by C-BRIC, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, the National Science Foundation, and Vannevar Bush Faculty Fellowship.

2575-8411/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDCS51616.2021.00061

from the edge network. This can minimize communication and resource usages for edge devices and maximize inference efficiency.

In addition to distributed inference, since IoT devices continuously collect new data from the environment, distributed training is challenging but beneficial. Although it is possible to upload all data to the cloud for training and then download the updated model, the communication energy and latency would be huge, and the large amount of IoT devices would put significant pressure on the cloud server to respond. Besides, the data privacy is a big concern. Hence, there is a need to explore efficient training algorithms for resource-constrained edge devices. If the model can be adaptively trained with locally collected data, it can better fit the environment, making it possible to outperform the pretrained models. Furthermore, the deployment of pretrained models puts intelligent IoT under the risk of white-box attacks because attackers may access the model parameters when the model is downloaded [4], [8]. Training at the edge modifies model parameters locally, and hence may prevent such attacks.

Due to the resource and energy constraints, it is infeasible to train with a large dataset and do backpropagation in a deep neural network at the edge. To overcome this and realize intelligent edge-cloud distribution, complexity-aware training strategies provide a way to selectively utilize data and parameters. The complexity of data varies widely across instances and classes in real-world datasets, which makes the classification difficulty not uniform [9]. For example, Fig. 2 shows the confusion matrix of the CIFAR-10 dataset. The precision of some classes is notably lower than others, which reflects their higher **class-wise complexity**. We classify data into three categories: easy, hard, and complex. Fig.3 shows the definition of these categories depending on the instance-wise and class-wise complexity. The complexity-aware training strategy aims at specifically improving the classification of hard classes and reducing the required amount of training data. On the other side, complexity-aware inference strategies ensure that instances with high **instance-wise complexity** are sent to the cloud for better accuracy. Early exiting of inference also ensures low latency and energy-efficiency.

In this paper, we design a distributed network architecture that is suitable to apply the proposed complexity-aware training and inference strategies. It consists of an adaptive convolutional neural network (CNN) at the edge and a deep CNN at the cloud. The architecture is shown in Fig. 1. The edge part is divided into a *main block*, an *extension block* and an *adaptive block*. The architecture leverages a pre-trained main block and extend it by two blocks to offer flexibility and adaptivity. We call this tripartite architecture an "MEA" structure and the corresponding network, MEANet. Any deep feed-forward network such as ResNet can be restructured into an MEANet. The main block and the extension block each contain a fully-connected classifier (*exit*). Each exit generates its own prediction $\hat{y}$.

As the proposed system combines adaptive MEANet with a deep CNN on the cloud, its training is distributed to the edge and the cloud. The main block of MEANet is trained on the cloud then downloaded to the edge and gives high confidence on inference accuracy of "easy instances". The classes with lower validation accuracy given by the main block are automatically put in the set of "hard classes". The main block trained with all classes ensures the stable performance of the MEANet. The other two blocks of MEANet focus on improving the accuracy of hard classes, and hence they are trained locally with data from hard classes. The main block being fixed, the adaptive block creates a short path for backpropagation and provides features associated with hard classes. With the adaptive block added in the propagation path, the extension block is not completely dependent on the main block, and hence it is able to learn better about hard classes. Using the tripartite architecture and the complexity-aware training strategies, the heavy computation and memory cost of deep backpropagation is reduced.

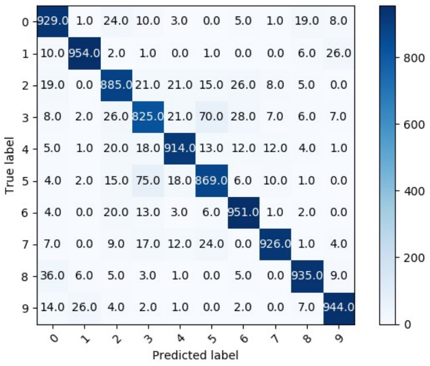

Fig. 2: Confusion matrix of running a ResNet32 on the CIFAR-10 dataset. The diagonal of the matrix indicates the number of correct classifications in each class.

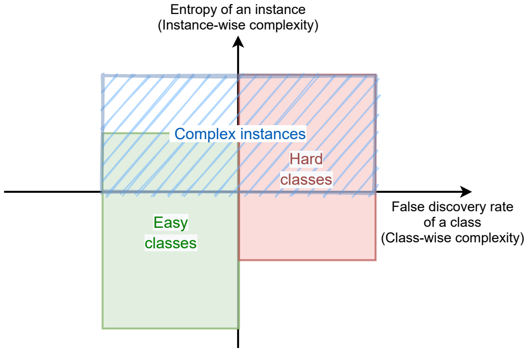

Fig. 3: Complexity of instances belonging to easy/hard/complex categories. The false discovery rate (FDR) of every class is evaluated by the main block on the validation set. FDR is the ratio of the number of false positive results to the number of total positive test results ($FDR = 1 - precision$). The entropy of an instance is evaluated by the main block during inference. Please note that the easy and hard categories have no overlaps, but an instance being either easy or hard may also belong to the complex category.

The inference process of MEANet also involves both the edge and the cloud based on the complexity-aware strategies. During inference, each exit controls the activation of the main block, extension block, or the cloud. For example, if an input is detected as an easy class with high probability, it can terminate immediately at the main block. An input detected as hard with high probability would be sent to the other blocks of the edge. If it does not show high probability of belonging to neither easy nor hard classes at the main block, then it is considered complex and is sent to the cloud. After further inference at the cloud, results are sent back to the edge. Note, the training algorithm and the network architecture lend to automatically determining the classes: easy, hard, and complex. While most instances of easy and hard classes are conditionally inferred at the edge, complex instances are detected at the edge but sent to the cloud for classification. Also note, that the edge device itself comes with conditional inference – the early exit for easy classes from the main block leads to energy efficient inference at the edge.

The main contributions of this paper are

- We propose a novel architecture, MEANet, that combines the pretrained main block with locally trained extension and adaptive blocks. MEANet at the edge, together with a DNN at the cloud, form a distributed system to smartly manage and process data.
- We propose complexity-aware methodologies for distributed training and inference. The distributed system considers both class-wise complexity and instance-wise complexity of data. The proposed methodologies are applicable to a wide range of deep learning models.
- We discuss and evaluate different ways of edge-cloud collaboration and their advantages. We compare edge-cloud distributed approaches with edge computing and cloud computing in terms of accuracy and energy consumption.

The rest of this paper is organized as follows. Section II summarizes the related works. Section III describes the details of the proposed model along with the training and inference algorithms. Then we discuss different ways of edge-cloud collaboration. Section III shows the implementation details and analyzes the experimental results. Conclusions are drawn in Section IV.

II. Related Work

A. DL Algorithms and Architectures for Edge Computing

To make deep learning available for edge/mobile computing, efficient algorithms and architectures are proposed, including quantization, pruning, branching, MobileNets, and neural architecture search. Quantization methods involve reducing the precision of weights and activations [7], [10]–[14]. Pruning is another technique [15]–[17] of reducing the number of parameters and computational complexity. Branching creates dynamic inference paths to conditionally activate layers according to the difficulty or property of the input, in order to improve efficiency or accuracy. Multi-branch neural networks, such as BranchyNet [6], use different training, early-exiting, and dynamic routing strategies [18]–[22]. MobileNets, which use depthwise separable convolutions, allow shrinking the model by global hyper-parameters to match the resource restrictions for specific applications [23] [24]. Neural architecture search first finds the best network architecture in a search space [25], then deploy it to the edge.

Several existing training algorithms apply the idea of divide-and-conquer to enable efficient training. Incremental learning and continual learning use a continuous learning process to accommodate previously unseen data and tasks [26]–[29]. To alleviate catastrophic forgetting in continual learning, different studies can be partitioned into architectural, functional, and structural approaches. For example, an episodic memory can be used to store a subset of observed data [30]. Supervised pretraining aims to modify deep learning models into simpler versions that are easier to train [31]. Before powerful GPUs became available, this was one of first attempts to train deep networks. Nevertheless, combining the optimal sub-models does not always produce an optimal deep learning model. Similarly, Bloctrain has been proposed to freeze trained blocks while gradually adding new blocks for training spiking neural networks [32].

B. Distributed Deep Learning

Distributed DL includes distributed inference and training for a multi-node system or a edge-cloud system. The main idea of distributed DL for the edge is to partition a deep neural network or coordinate multiple shallow neural networks. Layer-based partitioning of DNNs is a natural approach that applies branching, but distributing multiple branches at different nodes would result in transmitting a large amount of intermediate data. Another way to distribute a DNN is to divide each convolutional layer into independently distributable tasks, thus enabling parallel processing [33]. TeamNet provides a mixture-of-experts approach which trains multiple expert models with data selected by a gate network [34]. Large-scale distributed DL systems with multiple nodes can achieve high performance by parallelized optimization algorithms [35]–[39].

Besides improving computing efficiency and speed, the other aspect of distributed DL is edge-cloud collaboration. In [5] and [40], an intermediate layer with lightweight features is selected as a partition point between the edge and the cloud to save communication cost. In [41], a scheduler called Neurosurgeon is proposed to automatically partition DNN computation between mobile devices and datacenters at the granularity of layers. SPINN is a distributed inference system that co-optimizes the early-exit policy and DNN partition at runtime, in order to adapt to dynamic conditions [42]. Distributed DL can be combined with quantization techniques to build hybrid networks, which consist of low precision layers at the edge and full precision layers on the cloud [43]. Training can also be distributed across the edge and cloud. For example, gradient-descent based distributed learning algorithms involve local update steps at edge nodes and global aggregation steps performed by the cloud [44].

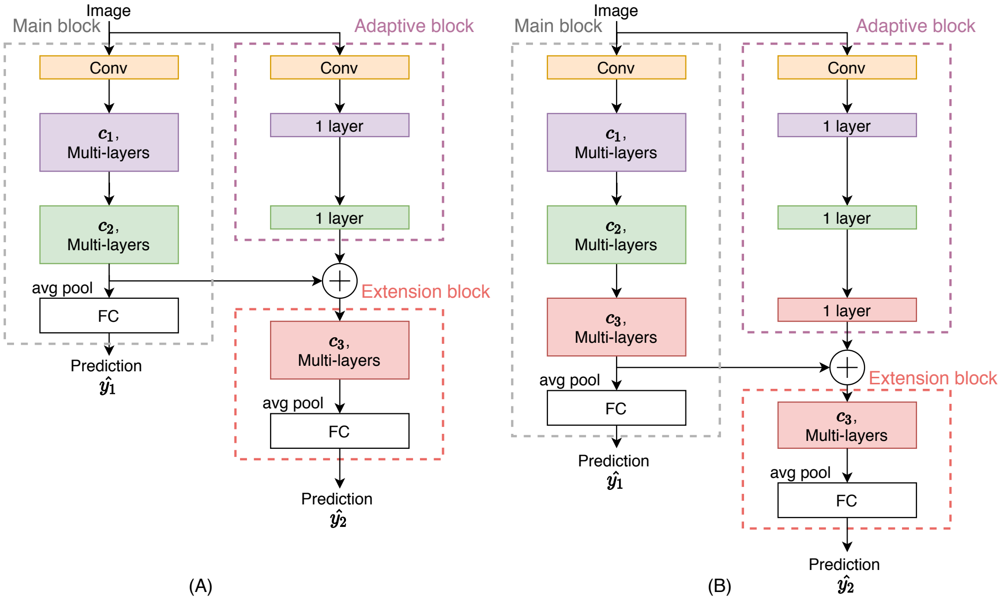

Fig. 4: Examples of MEANets. c_1, c_2, c_3 are the numbers of output channels of convolutional layers. Having the same color means having the same number of output channels and kernel size.

III. Complexity-aware adaptive neural networks

A. Training

The motivation for training at the edge is to improve the adaptivity of the edge device and to enhance its perception on data. However, we must consider the resource and energy constraints. Since backpropagation in the training process is computationally expensive, training a deep neural network at the edge is challenging. Also, it is impracticable to store the entire dataset at the edge or require the IoT devices to collect data of all classes. Furthermore, while having multiple exits enables complexity-aware learning, the optimization of these exits requires a special training algorithm. Hence, it is essential to design an adaptive architecture and a suitable training algorithm for the edge.

First, let us describe the architecture of the MEANet, which is shown in the bottom part of Fig. 1. The main block contains the majority of convolutional layers of the edge neural network. Because of the large number of parameters, it is preferred to train network in the cloud, then deploy to the edge. Having obtained a pretrained main block, the extension block is added to increase the depth of the network. It produces classification results $\hat{y_2}$ in addition to $\hat{y_1}$ from the main block. Although the main block is able to perform classification, it yields lower precision in hard classes. The extension block brings more layers to help making decision in hard classes. With the assistance of the extension block, MEANet has stronger learning ability to deal with complex data.

The purpose of using the adaptive block is to create a short path for backpropagation. The problem with training only the extension block is that it is strongly affected by the main block. Although we optimize the extension block for the loss at the local exit, the inputs to the block are features generated by the fixed main block. Thus, it is likely to perform the same misclassifications as the main block. It is important to make sure that the extension block is trained based on the raw inputs, which are independent of parameters at the main block. Hence, we use an adaptive block to connect the extension block with the raw inputs. The outputs of the adaptive block have the same size as those of the main block. Then the sum or concatenation of them are used as the inputs to the extension block. If concatenation is chosen, the first layer in the extension block would have more input channels. Compared to the main block, the adaptive block is designed to be much shallower to limit the corresponding computational and memory overhead.

Two examples of the MEANet are shown in Fig. 4. The adaptive block consists of convolutional layers similar to those in the main block. The number of out channels of each layer in the adaptive block matches with the corresponding layer in the main block. Therefore, their outputs have the same size. In other words, the adaptive block is a light-weight version of the main block. In model A, a typical CNN such as ResNet is divided into two parts. The first part is used as the main block and the second part including the fully connected layer is used as the extension block. An extra fully connected layer (exit) is created for the main block. In this case, the main block has fewer layers, so it is possible to also train it at the edge. In model B, a complete CNN is used as the main block, then the adaptive block and extension block are added. Model B has a deeper main block than the model A. Consequently, it generally achieves better accuracy, but its main block must be trained at the cloud before downloading to the edge. Note, the additional blocks can always be trained at the edge.

Algorithm 1: Distributed training of the edge neural network

```
Input: Instances $X$ and labels $Y$
Output: Trained neural networks
1. Train a deep CNN as the cloud AI. Also train the
   main block of the edge AI at the cloud with the
   whole dataset.
2. Run evaluation on the validation set to determine
   hard classes $C_{hard}$.
3. Create a dictionary ClassDict to map labels of all
   classes to new labels of hard classes.
label = 0;
for $c$ in $C$ do
    if $c \in C_{hard}$ then
        ClassDict[c]= label;
        label+=1;
    end
end
4. Download the main block and ClassDict to the edge.
5. Select training data $X_{hard}$ and generate new labels
   $Y_{hard}$.
for instance $i$ in $X$ do
    if $Y[i] \in C_{hard}$ then
        Index.append(i);
        $Y$[i]= ClassDict[$Y$[i]];
    end
end
$X_{hard}$ = X[Index]; $Y_{hard}$ = Y[Index];
6. Add the adaptive block and the extension block to
   the neural network. Fix the main block.
7. Forward propagation.
$\hat{y_1}, F =$ Main-block($X_{hard}$);
$f_2 =$ Adaptive-block($X_{hard}$);
$\hat{y_2} =$ Extension-block($F$,$f_2$);
8. Backward propagation.
Loss = CrossEntropyLoss($\hat{y_2}$, $Y_{hard}$);
Loss.backward();
```

Having described the adaptive architecture, we then propose a training algorithm for the MEANet. To save training efforts at the edge, the training algorithm reduces training data and uses blockwise optimization. There are three different methods to train multiple exits in a neural network: joint optimization, separate optimization, and blockwise optimization. Joint optimization optimizes the weighted sum of losses at all exits [6]. Separate optimization trains all convolutional layers based on the loss at the final exit, then freezes them and trains the other exits. Blockwise optimization divides the neural network into blocks. Each block contains several convolutional layers and an exit. When a block is trained, all blocks ahead of it are frozen, leading to improvement in training time and training energy. In our training algorithm, we use blockwise optimization for the following reasons. Although using joint optimization achieves the best accuracy, it is not suitable for training at the edge. The edge can hardly afford to train all parameters of a deep network at the same time. In order to address the challenge of training with resource constraints, blockwise optimization is preferred. It only requires to store gradients of the non-fixed parts for backpropagation. Consequently, it largely reduces the memory requirement and energy consumption. Moreover, by fixing the trained main block, it avoids unnecessary training efforts and secures the learned properties.

The detailed steps of the distributed training algorithm are described in Algorithm 1. First, the main block is trained at the cloud with the entire dataset. Next, it is run on the validation dataset to produce class-level statistics. Then we rank the classes according to their precision in increasing order and define first N_{hard} classes as hard classes, where N_{hard} is a user-defined parameter. Other classes are defined as easy classes. Because the labels of hard classes are not likely to be consecutive in the set of all classes C, we generate a new set of labels exclusively for hard classes and call it C_{hard}. Then the set of easy classes is given by $C - C_{hard}$. A dictionary is created to record the mapping of C and C_{hard}. After that, we download the main block to the edge and train the extension and adaptive blocks. Since the main block has been trained, it is fixed while training the other blocks. The extension and adaptive blocks aim to deal with complex cases, therefore, it is unnecessary to train them on the entire dataset. Instead, we only use instances in hard classes. The labels of instances are checked and only those belonging to C_{hard} are used for training. As a result, the number of classes is reduced and the size of training dataset that is usually proportional to the number of classes becomes smaller. More importantly, the weights of the adaptive block provide features specifically for hard classes. By learning from these features, the extension block is able to correct the decision boundaries suggested by features from the main block. The remaining question is how to detect instances of hard classes and manipulate the activation of these blocks during inference. We will describe this in the next subsection.

Please note that in our experiments, we use the hard-class samples from the training dataset to train the extension and the adaptive blocks. This simulates the case that data collected from the environment have the same distribution as those in the dataset. In the real environment, the edge can collect new samples that have a different distribution. To avoid overfitting and catastrophic forgetting on the new samples, we suggest using both the new samples and samples from the dataset for training.

B. Inference

During inference, whether an instance belongs to one of the hard classes or easy ones is decided based on the prediction by the main block. Although it is optional to train a binary classifier as a detector, we find that using the outputs of the main block to detect easy/hard classes is the simplest and the most effective way. The complexity-aware partition of classes ensures accurate detection since the average precision on easy

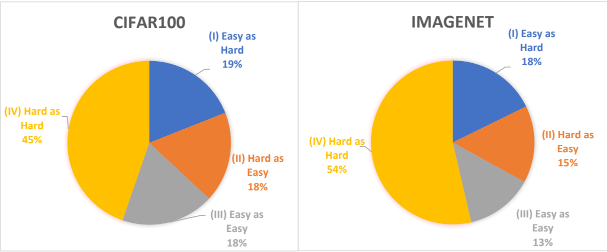

Fig. 5: The proportion of four types of errors: (I) mistaking a sample of a easy class as hard; (II) mistaking hard as easy; (III) mistaking a sample of a easy class as another easy class; (IV) mistaking a sample of a hard class as another hard class. Here we assume half of classes are hard.

classes is higher than that on all classes. We denote the output vector of the main block by $\hat{y_1}$, whose length equals the number of classes. Then the softmax function is applied to get the probability p_1 of belonging to each class. The predicted class is the index of the maximal element in p_1. If the predicted class is one of the easy classes, the corresponding instance is classified as belonging to easy classes and is allowed to exit at the main block; otherwise, the other two blocks are activated. The detection of easy/hard classes can be summarized as

$$p_1 = softmax(\hat{y_1})$$
$$IsHard(\hat{y_1}) = \begin{cases} True, & \text{if } argmax(p_1) \in C_{hard} \\ False, & \text{otherwise} \end{cases}$$

The detection of easy class vs. hard class is not 100% correct, however, the misinformation is not fatal due to the following reasons. Fig. 5 depicts the proportions of four types of errors when evaluating the main block on CIFAR100 and ImageNet. The four types are (I) mistaking a sample of a easy class as hard; (II) mistaking hard as easy; (III) mistaking a sample of a easy class as another easy class; (IV) mistaking a sample of a hard class as another hard class. Around half of the errors by the main block belong to type IV. The extension block trained on hard data can improve type IV errors. As long as the extension block makes considerable improvement on this type, the overall accuracy will increase accordingly. If the IsHard function is modified to consider more samples as hard, some type II errors will be turned into type IV, which may take more advantage of the extension block. However, adjusting the detection function would also cause sending instances of easy classes to the extension block by mistake. In such cases, the extension block is not able to classify these instances correctly since it is trained with only hard classes. Therefore, modifying the IsHard function can not guarantee an overall improvement. Nevertheless, if the main block confuses easy classes with hard ones, then its prediction is wrong and the prediction by the extension block could not be worse. Hence, although the imperfect detection limits the improvement brought by the extension block, it is acceptable.

After the extension block provides new predictions, we compare predictions from the two exits at the main block and the extension block and select the one with higher confidence. The softmax score of the predicted class, which is also the maximal softmax score among all classes, indicates the confidence on the prediction. This is to ensure the final prediction improves compared to the prediction by the main block. Furthermore, the wrongly detected instances belonging to complex instances can be handled by the cloud. We will describe the details in the next subsection.

C. Distributed inference by edge and cloud

The edge model can either infer data independently, which ensures data security, or ask a deeper CNN at the cloud to deal with complex instances. The design methodology for MEANet improves the accuracy of hard classes, however, there are complex instances in both easy and hard classes. Hence, both the extension block and the cloud are needed to deal with class-wise complexity and instance-wise complexity, respectively. Making edge and cloud collaborate for inference can improve the tradeoff between energy and accuracy by exploiting edge resources and occasionally activating cloud resources. Note, it is important to utilize edge resources to analyze data locally as much as possible for faster inference. Although the deeper network at the cloud is more accurate, a large portion of instances might have been predicted correctly at the edge. Therefore, sending them to the cloud is a waste of energy and time. Furthermore, having the option of sending to the cloud solves the misdetection of easy/hard classes. Since the misdetected instances do not belong to any classes at the extension block, they should be directly sent to the cloud.

The entropies of predictions at the main block are used to decide which instances to send to the cloud, similar to that in [5], [43]. The entropy of a prediction indicates the instance complexity and the confidence of the classifier. At the main block, the entropy values of correct ones show an exponential distribution peaking at zero, while those of wrong predictions show a normal distribution whose mean is larger than one. Instances with entropy higher than a threshold are sent to the cloud. By evaluating the entropy values of the validation set, the range of the threshold can be determined as (μ_c, μ_w). Here μ_c is the mean of entropy of correct predictions and μ_w is that of wrong predictions. Then the user can select a threshold in the range based on the system requirements. The higher the threshold is, fewer instances are sent to the cloud and the inference is faster and more efficient.

There are two ways of edge-cloud distributed inference. Either raw data or processed features are sent to the cloud depending on the architecture of the cloud network. In many distributed learning work, features are sent because the layers at the edge and cloud are components of a partitioned deep neural network [5]. When the component at the cloud receives output features from the edge as inputs, it continues processing and sends back results. This approach seems natural, but sending features forces the data processing at the cloud to be dependent on the data processing results from the edge. On the other hand, if raw data are given, the cloud AI can be independent of the edge AI. Hence, a more accurate network

	Edge computation	Cloud computation	Communication	Data Privacy
Edge	$N \cdot x$	/	/	++++
Cloud	/	$N \cdot x_{cl}$	$N \cdot x_{cu}$	+
Edge-cloud (sending raw data)	$N \cdot x$	$\beta \cdot N \cdot x_{cl}$	$\beta \cdot N \cdot x_{cu}$	++
Edge-cloud (sending features)	$N \cdot (qx)$	$\beta \cdot N \cdot (1-q)x_{cl}$	$\beta \cdot N \cdot x'_{cu}$	+++

TABLE I: Cost estimation of inference using edge, cloud and edge-cloud collaboration. N is the total number of data instances. x is the energy consumption or the latency to process an instance at the edge. β is the percentage of data sent to the cloud. x_{cl} is the cost of cloud computation. x_{cu}, x'_{cu} are the cost of communication. q is the proportion of layers distributed at the edge.

can be used at the cloud to improve the accuracy and the edge-cloud system is more flexible. For this reason, we use a ResNet101 as the cloud network and send raw data to the cloud.

Algorithm 2 shows the overall process of inference in the edge-cloud distributed system. In summary, all instances are first processed by the main block. Then instances with high entropy (low confidence) are sent to the cloud. Other instances are allowed to exit at the main block if they are classified as belonging to easy classes; otherwise, they are sent to the extension block.

Algorithm 2: Inference in an edge-cloud AI system

```
Input: A batch of images $X$ from the dataset
Output: Classification results $\hat{Y}$, features $F$
for a batch of instances do
    $\hat{y_1}, F$ = Main-block($X$);
    for instance i in $X$ do
        if Cloud is available and Entropy($\hat{y_1}$[i]) > threshold then
            Send instance I[i] or feature $F$[i] to the cloud;
            Receive the classification result;
            $\hat{Y}$[i] = $\hat{y_{cloud}}$[i];
            continue;
        end
        if IsHard($\hat{y_1}$[i]) then
            $f_2$ = Adaptive-block($X$[i]);
            $\hat{y_2}$[i] = Extension-block($F$[i]+$f_2$);
            if Confidence($\hat{y_1}$[i]) > Confidence($\hat{y_2}$[i]) then
                $\hat{Y}$[i] = $\hat{y_1}$[i];
            else
                $\hat{Y}$[i] = $\hat{y_2}$[i];
            end
        else
            $\hat{Y}$[i] = $\hat{y_1}$[i];
            continue;
        end
    end
end
```

D. Cost estimation for inference

In Table. 1, the cost of edge-cloud distributed inference is compared with that of edge inference or cloud inference. The cost can be energy consumption or latency. Using this table, one can estimate the energy or latency of inference by inserting values of variables. Total cost is broken down into the computation cost of the edge, the computation cost of the cloud, and the communication cost. Assume that the total number of data instances is N. The computation cost to process an instance at the edge and that at the cloud are x and x_{cl}. Then the total cost of edge computation is $N \cdot x$. The communication cost can be either x_{cu} or x'_{cu} corresponding to sending raw data or features respectively. Whether sending raw data or features consumes more communication energy depends on the dataset. If the size of data in a dataset is small (e.g. CIFAR datasets), the size of features is usually larger than that of raw data. In the ImageNet dataset, the size of raw data might be larger. With early exiting allowed by edge-cloud collaboration, the percentage of data instances sent to the cloud is reduced to β in the range of $[0, 1]$. In the case of distributing layers and sending features to the cloud, the proportion q of layers distributed at the edge needs to be considered (typically $q \in [1/3, 2/3]$). In the next section, we will estimate energy consumption using this table.

IV. Experimental results

A. Setup

To validate the effectiveness of our methods, we conduct several sets of experiments with different CNN architectures and datasets. We use ResNet [2] and MobileNetV2 [24] as the basic architectures and extend them as described in section II. As image classification being one of the most popular computer vision tasks, we test the models on two image classification datasets, CIFAR-100 [45] and ImageNet [46].

We build a simulation environment using PyTorch to simulate edge-cloud distributed AI systems. The statistics including the overall accuracy, the accuracy of hard classes, and the percentage of instances exiting at each exit are recorded. Then the energy consumption for communication and computation is estimated.

The architectures of the models are illustrated in Fig.4. Because CIFAR dataset contains much fewer classes and images, the convolutional layers of ResNet used for CIFAR100 have 16, 32, and 64 channels respectively from the first group of layers to the last one ($c_1 = 16, c_2 = 32, c_3 = 64$). Every group contains the same number of convolutional layers. The small number of channels used for CIFAR100 makes it suitable for smart IoT edge. As described in section II, model A uses the a part of the original ResNet as the extension block, while model B keeps the complete ResNet as the main block and add new layers to be the extension block. After modification, we add the letter A or B in the name of the ResNet to show which type is used.

The ResNet used for ImageNet has four different groups of convolutional layers – 64, 128, 256, and 512 channels respectively. A pretrained ResNet18 is downloaded from PyTorch official website as the main block, then the adaptive and extension blocks are added and trained to change it into model B. We also apply the proposed method (model B) to MobileNetV2. Similar to what we do to ResNet, MobilenetV2 pretrained on the ImageNet dataset is used as the main block. The other blocks are subsequently added. To limit the computational overhead, the extension block for model B is designed to have four residual blocks.

For both datasets, 10% of the training data are used as the validation set to determine hard classes. The proportion of the training set used to train the extension and adaptive blocks depends on the number of hard classes. The default number of hard classes N_{hard} is half of the total number of classes, and consequently half of the training set is used. After training the main block or downloading the pretrained model, we add the adaptive and extension blocks to the model. In every training epoch, the model is set to train mode, while the layers in the main block are set to evaluation mode, and the corresponding parameters are set not to require gradients. Please refer to Section III for the detailed training steps. For CIFAR-100 experiments, the models are trained with initial learning rate equaling 0.1, then multiplied with 0.1 at epoch 60, 120, and 160. For ImageNet experiments, the models are trained with initial learning rate equaling 0.01, then multiplied with 0.1 at epoch 30 and 100.

B. Results

Dataset, model	Train		Test	
	main	MEANet	main	MEANet
CIFAR-100, ResNet32 A	69.84	97.22	50.70	59.76
CIFAR-100, ResNet32 B	77.95	96.07	59.36	63.66
ImageNet, MobileNetV2 B	65.25	70.53	61.26	65.26
ImageNet, ResNet18 B	61.01	72.33	59.98	64.95

TABLE II: Accuracy of hard classes (%)

1) Results on hard classes (edge): Because the learning process at the edge only uses training data belonging to hard classes, it is reasonable to first look at the training and testing accuracy of hard classes before and after adding the extension and adaptive blocks. In the tables, we call the results corresponding to the main block as "main" and those corresponding to MEANet as "MEANet". The results in Table. II are evaluated using only half of instances in the datasets which belong to hard classes. This simulates the case that the edge can only get data in these classes from the environment. Under this circumstance, the extension and adaptive blocks are always activated. The training accuracy of hard classes is significantly increased after adding the extension and adaptive blocks. Compared to the original model, the test accuracy of ours increases by 4-9% for CIFAR-100 and 4-5% for ImageNet, dependent on different architectures. Because the main block in model A has fewer layers compared to that in model B, the extension block is able to provide larger accuracy improvement than model B. The results show that our proposed model has a stronger ability to distinguish between hard classes.

Dataset, model	main	MEANet	easy/hard detection
CIFAR-100, ResNet32 A	61.70	63.51	83.47
CIFAR-100, ResNet32 B	67.55	67.80	87.88
ImageNet, MobileNetV2 B	71.87	73.19	90.64
ImageNet, ResNet18 B	69.75	71.62	88.49

TABLE III: Test accuracy(%) of all classes

2) Results on all classes (edge): Table. III lists the accuracy of the MEANet when the entire test dataset is evaluated. The overall accuracy on ImageNet increases by nearly 2% after adaptively training the MEANet. For the CIFAR dataset, Model A provides a larger improvement than model B because its main block is shallower and underfits the dataset. There are two reasons for the improvement being insignificant compared to the results on hard classes. First, because our model focuses on increasing the accuracy of hard classes, the increase is evened out when all classes are presented. Second, the misdetection of easy/hard classes by the main block weakens the strength of the extension block. To compensate for this, edge-cloud collaboration can be used. In conclusion, the advantage of our model is maximized if the main block underfits the dataset or the instances of the hard classes make up a large proportion of the dataset.

Selected classes	Detection accuracy(%)
50 hard	83.47
50 random	81.77
70 hard	86.85

TABLE IV: Detection accuracy of easy/hard classes on CIFAR-100

Selected classes	Training accuracy(%)		Test accuracy(%)	
	main	MEANet	main	MEANet
50 hard	69.84	97.22	50.70	59.76
50 random	76.51	98.42	61.42	69.64
70 hard	72.70	94.43	54.53	57.50
100	79.20	92.65	61.70	62.37

TABLE V: Effect of selection of classes on the accuracy of selected classes of CIFAR-100 dataset, using ResNet32 A

3) Effect of class selection: To analyze the effect of which classes to select for training, we compare selection based on class-wise complexity with random selection. In addition, we investigate the effect of the number of selected classes. Experiments are run with different numbers of selected classes including 50, 70, 100 out of 100 classes in CIFAR-100. Table.IV shows that the accuracy of detecting whether an instance belongs to C_{hard} is lower if randomized selection is used. Table.V indicates that as the number of selected classes increases, the improvement on training and test accuracy decreases although more training data are used. This is because our model benefits from the reduction in the number of classes, which makes it easier to find the decision boundaries. In conclusion, selecting based on class-wise complexity and defining a small set of hard classes are suggested to guarantee the advantage of our model.

4) Training cost: We then estimate the training cost of the proposed model. Because the main block can be pre-trained at the cloud, we only consider the training cost of other blocks at the edge, and compare our approach with joint optimization using data of all classes. Computation cost of training is composed of forward propagation cost and backward propagation cost. Both fixed parameters and trained parameters are used in the forward pass, therefore, this part of cost is the same for our approach and the joint approach. However, fixed parameters do not require gradient computation, and they do not cause any backpropagation cost. Since our approach fix the main block, only parameters in the adaptive block and the extension block are trained. In Table.VI, we show the numbers of parameters that are fixed or trained when using our approach. Also, we count the related number of computations in terms of Multiply-Adds using a Python package called ptflops [47]. The number of parameters and the corresponding number of computations directly affect the memory and computation cost. While our approach uses fixed parameters only for forward propagation, joint optimization [6] trains all the parameters together. Hence the number of computation related to fixed parameters equals to the reduced number of gradient computation of our approach compared to joint optimization. When both approaches train the model with the same batch size, our training algorithm uses 60% less GPU memory for ResNets and 30% less for MobileNets, as shown in Fig. 6. Furthermore, because we only use instances belonging to hard classes for training, the amount of training data is reduced by half. As a result, the total training cost to process the training dataset can be further reduced by half.

Dataset, model	# of computation		# of parameters	
	fixed	trained	fixed	trained
CIFAR-100, ResNet32 A	46	31	0.11	0.37
CIFAR-100, ResNet32 B	69	31	0.47	0.42
ImageNet, MobileNetV2 B	300	130	3.49	1.09
ImageNet, ResNet18 B	1722	2058	11.16	27.46

TABLE VI: Number of compuatation and number of parameters (Million)

5) Distributed inference by edge and cloud: We compare the accuracy between edge-only, cloud-only, and edge-cloud AI systems. Since the cloud server does not have a resource constraint, we use ResNet101 as the cloud AI. In Fig.7, it shows the overall accuracy and the percentage of data sent to the cloud depending on the threshold. When the threshold is zero, all data are sent to the cloud. Compared to the edge-only inference, distributed inference can improve the accuracy on CIFAR-100 by 2% by sending 15% of data to the cloud. The accuracy improvement on ImageNet is 4% when sending 28% of data. If a lower threshold is set, the percentage of data sent to the cloud and the accuracy would both increase. When the threshold is low enough (0.5), distributed inference can achieve similar accuracy as using only the cloud.

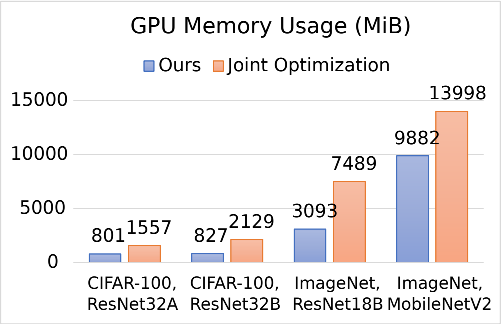

Fig. 6: GPU memory usage for training the extension and adaptive blocks with our algorithm or joint optimization (batch size = 128)

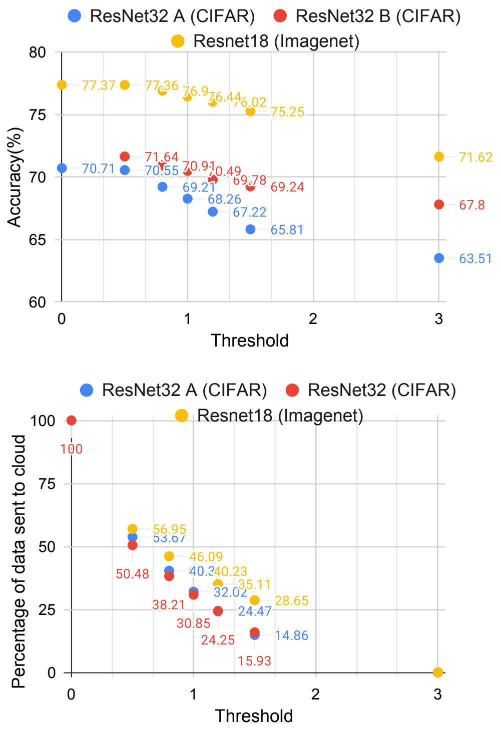

Fig. 7: Distributed inference by edge and cloud

Next, to compare the inference energy between edge-only, cloud-only, and edge-cloud AI systems, we calculate the energy consumption at edge, which is the sum of computational and communication energy of the edge. The computation energy of the cloud AI is ignored because it is not a concern. First, we calculate the communication energy to upload an image. Based on the power model of WiFi in [48] and [40], the power of uploading data is calculated by $P_{upload} = 283.17(mW/Mbps) \times s_{upload} + 132.86(mW)$, where s_{upload}

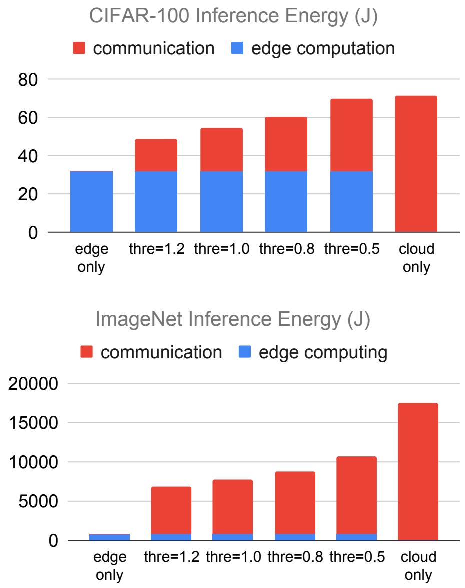

Fig. 8: Total energy consumption at the edge to infer 10000 images in CIFAR-100 testset using ResNet32 A; total energy consumption at the edge to infer 50000 images in ImageNet testset using ResNet18 B

is the throughput. We assume that the throughput equals to the average upload speed 18.88Mb/s. Then communication power is $P_{upload} = 5.48W$. Since the image size in CIFAR-100 is 32x32x3 (bytes) and the image size in ImageNet is 224x224x3 (bytes), the communication time t_{cu} to upload an image in CIFAR-100 or ImageNet is 1.3 ms or 63.7 ms respectively. Then the communication energy is calculated by $E_{cu} = P_{upload} * t_{cu}$.

Second, we estimate the computation energy and latency of the neural network at the edge. The GPU power is monitored by Nvidia system monitor. The latency for inferring an instance is the latency of GPU running a batch of instances divided by the batch size. Then the inference energy is calculated and listed in Table.VII. Please note that the GPU we use for simulating an edge-cloud system is Nvidia GeForce GTX 1080 Ti. This GPU is not designed for the edge. In a real edge-cloud system, a less powerful GPU can run the neural network at the edge with smaller batch size and may have longer latency and lower power.

In Fig.8, we show the total energy consumed by inferring all images in the test set of CIFAR-100 and ImageNet. We insert numbers to equations in Table. I to calculate the total energy. It shows that when the threshold is 0.5 for CIFAR-100, the total energy consumption at the edge is close to that of sending all samples to the cloud. However, since more than 50% of data inference have terminated at the edge, edge-cloud distributed inference still has the advantage in latency. In the case of ImageNet, the computation energy at the edge is much smaller than the communication energy because of the large image size. Hence, the distributed inference achieves the same ImageNet accuracy as the cloud by consuming only 60% of energy at the edge.

V. Conclusion

With energy and resource constraints at the edge, smart IoT systems are facing great difficulty to process data promptly and accurately. In this work, we propose a distributed system that lends to energy-efficient and fast classification of most instances on the edge device, while activating the cloud for complex ones on rare occasions. The complexity of an instance is evaluated by the edge using the proposed MEANet. The energy efficiency of the edge is due to conditional inference of easy and hard classes using early exits at the main or the extension block. Instances that can not be identified as easy or hard classes by the main block are considered "complex" and sent to the cloud. The proposed architecture, training and inference techniques are applicable to typical CNNs such as ResNets and MobileNets. The experimental results show that the proposed model can obtain higher accuracy at a low training cost, which confirms its potential for application to real-life edge devices. With edge-cloud collaboration, the system achieves a better accuracy-vs-energy tradeoff compared to cloud-only or edge-only approaches.

References

[1] Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] He *et al.*, "Deep residual learning for image recognition," in *Proceedings of CVPR*, pp. 770–778, 2016.

[3] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4921–4934, 2019.

[4] M. Shafique, T. Theocharides, C. Bouganis, M. Hanif, F. Khalid, R. Hafiz, and S. Rehman, "An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the iot era," 03 2018.

[5] S. Teerapittayanon *et al.*, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 328–339, June 2017.

[6] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, 2016.

[7] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.

[8] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," *CoRR*, vol. abs/1602.02697, 2016.

[9] P. Panda, A. Sengupta, and K. Roy, "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 475–480, 2016.

[10] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of CVPR*, June 2018.

[11] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *ECCV*, vol. abs/1603.05279, 2016.

[12] S. Zhou *et al.*, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

Dataset, model	GPU power(W)	WIFI uploading power (W)	t_{cp} (ms)	t_{cu}(ms)	E_{cp}(mJ)	E_{cu}(mJ)
CIFAR-100 ResNet32 A	56	5.48	0.056	1.3	3.14	7.12
ImageNet ResNet18 B	75	5.48	0.203	63.7	15.23	349

TABLE VII: Computation and communication power, time, and energy at the edge per image

[13] I. Chakraborty, D. Roy, I. Garg, A. Ankit, and K. Roy, "Constructing energy-efficient mixed-precision neural networks through principal component analysis for edge intelligence," *Nature Machine Intelligence*, pp. 1–13, 2020.

[14] J. Choi *et al.*, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.

[15] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[17] I. Garg, P. Panda, and K. Roy, "A low effort approach to structured cnn design using pca," *IEEE Access*, 2019.

[18] Y. Tang, Y. Wang, H. Li, and X.-W. Li, "Mv-net: Toward real-time deep learning on mobile gpgpu systems," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 15, pp. 1–25, 10 2019.

[19] M. McGill and P. Perona, "Deciding how to decide: Dynamic routing in artificial neural networks," in *Proceedings of the 34th International Conference on Machine Learning*, pp. 2363–2372, 2017.

[20] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "Blockdrop: Dynamic inference paths in residual networks," in *Proceedings of CVPR*, pp. 8817–8826, 2018.

[21] T. G. Xin Dai, Xiangnan Kong, "Epnet: Learning to exit with flexible multi-branch network," p. 235–244, October 2020.

[22] H. Li, H. Zhang, X. Qi, Y. Ruigang, and G. Huang, "Improved techniques for training adaptive deep networks," in *2019 IEEE/CVF International Conference on Computer Vision*, pp. 1891–1900, 2019.

[23] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[24] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of CVPR*, June 2018.

[25] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *CoRR*, vol. abs/1707.07012, 2017.

[26] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of CVPR*, July 2017.

[27] S. S. Sarwar, A. Ankit, and K. Roy, "Incremental learning in deep convolutional neural networks using partial network sharing," *IEEE Access*, vol. 8, pp. 4615–4628, 2020.

[28] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70 of *Proceedings of Machine Learning Research*, pp. 3987–3995, PMLR, Aug 2017.

[29] J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell, "Progress & compress: A scalable framework for continual learning," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 4528–4537, PMLR, 10–15 Jul 2018.

[30] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continuum learning," *NIPS*, vol. abs/1706.08840, 2017.

[31] L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, *Scaling Learning Algorithms toward AI*, pp. 321–359. 2007.

[32] G. Srinivasan and K. Roy, "Bloctrain: Block-wise conditional training and inference for efficientspike-based deep learning," 2020.

[33] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[34] Y. Fang, Z. Jin, and R. Zheng, "Teamnet: A collaborative inference framework on the edge," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1487–1496, 2019.

[35] D. Li, Z. Lai, K. Ge, Y. Zhang, Z. Zhang, Q. Wang, and H. Wang, "Hpdl: Towards a general framework for high-performance distributed deep learning," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1742–1753, 2019.

[36] M. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized stochastic gradient descent.," vol. 23, pp. 2595–2603, 01 2010.

[37] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, p. 583–598, 2014.

[38] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," NIPS'12, (Red Hook, NY, USA), p. 1223–1231, Curran Associates Inc., 2012.

[39] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," *CoRR*, vol. abs/1807.11205, 2018.

[40] A. E. Eshratifar and M. Pedram, "Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, p. 111–116, 2018.

[41] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 615–629, 04 2017.

[42] S. Laskaridis, S. Venieris, M. Almeida, I. Leontiadis, and N. Lane, "Spinn: synergistic progressive inference of neural networks over device and cloud," pp. 1–15, 09 2020.

[43] Y. Long, I. Chakraborty, and K. Roy, "Conditionally deep hybrid neural networks across edge and cloud," *arXiv preprint arXiv:*, 2020.

[44] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 63–71, 2018.

[45] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[46] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[47] V. Sovrasov, "Flops counter for convolutional networks in pytorch framework," 2020.

[48] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4g lte networks," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, (New York, NY, USA), p. 225–238, Association for Computing Machinery, 2012.