Distributed Deep Neural Networks 
over the Cloud, the Edge and End Devices

I. INTRODUCTION

Neural networks (NNs), and deep neural networks (DNNs) in particular, have achieved great success in numerous applications in recent years. For example, deep Convolutional Neural Networks (CNNs) continuously achieve state-of-the-art performances on various tasks in computer vision and natural language processing as shown in Figure 1. At the same time, the number of end devices, including Internet of Things (IoT) devices, has increased dramatically. These devices make appealing targets for machine learning applications as they are often connected to sensors (e.g., cameras, microphones, gyroscopes) that capture a large quantity of input data in a streaming fashion.

However, the current state of machine learning systems on end devices leaves an unsatisfactory choice: either (1) offload input sensor data to large NN models (e.g., DNNs) in the cloud, with the associated communication costs and privacy issues, or (2) perform classification directly on the end device using simple NNs or methods (e.g., linear SVM), leading to reduced system accuracy.

To address this concern, it is natural to consider the use of a distributed computing approach. Hierarchical distributed computing structures consisting of the cloud, the edge and devices have inherent advantages, such as supporting coordinated central and local decisions, allowing geographical span, and providing system scalability, for large-scale intelligent tasks based on geographically distributed IoT devices. These attributes are fundamental in provisioning large-scale DNNs to support such tasks.

For example, we could adopt a distributed approach that combines a small NN method on end devices and a large method in the cloud. The small method at an end device can quickly perform initial feature extraction, and also classification if the method is confident. Otherwise, the end device can fall back to a large NN method in the cloud, which performs further processing and final classification. This distributed approach has the benefit of low communication costs compared to always offloading input sensor data to a large NN model in the cloud and can achieve higher accuracy compared to a simple model on device. Additionally, since features from the device model are sent instead of input sensor data, the system provides better privacy by default.

However, distributing DNNs over computing hierarchies is challenging for a number of reasons, including:

- End devices such as embedded sensor nodes often have very little memory and limited battery budget so even if a small NN can be trained to fit the available resources, the achieved accuracy may not be acceptable for the application.
- A straightforward partitioning of a DNN over a computing hierarchy, either across multiple layers or by sections of a single layer may incur prohibitively large communication costs in transferring intermediate results between computation nodes.
- Incorporating geographically distributed end devices is generally beyond the scope of DNN literature. When multiple sensor inputs on different end devices are used, they need to be aggregated together for a single classification objective.
- Multiple models at the cloud, the edge and the device need to be learned jointly to allow coordinated decision making. Additionally, computation already performed on end device models should be useful for further processing on edge or cloud models.

The term network layer may refer to either a layer in a NN or a layer in the distributed computing hierarchy (e.g., edge or cloud). In order to remove ambiguity, when we refer to network layers for NN we explicitly use the term NN layers.
• Usual layer-by-layer processing of a DNN from the input layer all the way to the output layer does not provide a mechanism for local and fast inference at earlier points in the neural networks (e.g., end devices). In this case, we need to automate the sending of input samples to the edge or the cloud for further processing.

• A balance is needed between the accuracy of a model (with the associated model size) at a given distributed computing layer and the cost of communicating to the layer above it. The trained DNN needs to have reasonably good lower NN layers on the end devices capable of accurate local classification while also providing useful features for classification in the cloud.

Therefore, it is desirable that a system could train a single end-to-end model, such as a DNN, and partition it between end devices and the cloud, in order to provide a simpler and more principled approach.

To this end, we propose distributed deep neural networks (DDNNs) over distributed computing hierarchies, consisting of the cloud, the edge (fog) and geographically distributed end devices. In implementing a DDNN, we map sections of a DNN onto a distributed computing hierarchy. We show that DDNNs can effectively address all the aforementioned challenges. Specifically, while being able to accommodate inference of a DNN in the cloud, a DDNN allows fast and localized inference using some shallow portions of the DNN at the edge and end devices. Moreover, via distributed computing, DDNNs naturally enhance data privacy and system fault tolerance for DNN applications. When supported by a scalable distributed computing hierarchy, a DDNN can scale up in neural network size and scale out in geographical span. DDNN relies on the recent work of binary neural networks (BNNs) [3], which greatly reduce the required memory cost of neural network layers and enables multi-layer NNs to run on end devices with small memory footprints [10]. DDNN places multiple exit points in the DNN, by extending recent work on BranchyNet to the distributed computing scenario [15], which allows samples to be classified and exited locally when the system is confident and offloaded to the edge and the cloud when additional processing is required. By training DDNN end-to-end, DDNN optimally configures lower NN layers to support local inference at end devices, and higher NN layers in the cloud to improve overall classification accuracy of the system. As a proof of concept, we show a DDNN can exploit geographical diversity of sensors (on a multi-view multi-camera dataset) to improve recognition accuracy.

II. RELATED WORK

In this section, we cover the distributed computing hierarchy and the recent deep network algorithms that enables our proposed method to run in a distributed fashion. We then discuss other approaches involving distributed deep networks.

A. Distributed Computing Hierarchy

The framework of large-scale distributed computing hierarchy has assumed new significance in the emerging era of IoT. It is inevitable that most of data generated by massive IoT devices must be processed locally at the devices or at the edge, for otherwise the total amount of sensor data for a centralized cloud would overwhelm the communication network bandwidth. In addition, distributed computing hierarchy offers opportunities for data security and privacy, as well as short response times; these advantages have been eloquently articulated in articles [13, 16]. In [16], a face recognition experiment shows a reduced response time is achieved when a smartphone’s photos are proceeded by the fog (edge) as opposed to the cloud. In this paper, we show that DDNN can systematically exploit the advantages of a distributed computing hierarchy for DNN applications.

B. Deep Network Extensions

Binarized neural networks (BNNs) are a recent type of neural network, where the weights in linear and convolutional layers are constrained to \{−1, 1\} (stored as 0 and 1 respectively). This representation has been shown to achieve similar classification accuracy when compared to a standard floating-point neural network while using less memory space and reduced computation due to the compact binary format [3]. These compact models are especially attractive in end device settings, where memory can be a limiting factor. Embedded binarized neural networks (eBNNs) extends BNNs to allow the network to fit on embedded devices by reducing floating-point temporary through reordering the operations in inference [10]. In DDNN, we use binary convolutional and binary linear eBNN layers to accommodate the end devices, so that they can be jointly trained with the network layers in the edge and cloud.
BranchyNet proposed a solution of classifying samples at earlier points in a neural network, called exit points, through the use of an entropy-based confidence criteria [15]. If a sample is deemed confident based on the entropy of the computed probability vector for target classes, then it is classified and no further work is performed by the higher NN layers. In DDNN, exit points are placed at physical boundaries (e.g., between the last layer on an end device and first layer in the edge). Input samples that can already be classified early will exit locally, thereby saving the communication over the next physical boundary. Similar to BranchyNet, SACT [6] allocates computation on a per region basis in an image, and exits each region independently when it is deemed to be of sufficient quality.

C. Distributed Training of Deep Networks

Current research on distributing deep networks is mainly focused on improving the runtime of training the network. In 2012, Dean et al. proposed DistBelief, which maps large DNNs over thousands of CPU cores during training [5]. More recently, several methods have been proposed to scale up DNN training across GPU clusters [7, 4], which further reduces the runtime of network training. Note that this form of distributing DNNs (over homogeneous cores) is fundamentally different from the notion presented in this paper. We propose a way to train and perform feedforward inference over deep networks that can be deployed over a distributed computing hierarchy, rather than processed in parallel over bus- or switch-connected CPUs or GPUs in the cloud.

III. Proposed Distributed Deep Neural Networks

In this section we give an overview of the proposed distributed deep neural network (DDNN) architecture and demonstrate how training and inference in DDNN is performed.

A. DDNN Architecture

DDNN maps a trained DNN onto heterogeneous physical devices distributed locally, at the edge, and in the cloud. Since DDNN relies on a unified DNN framework at all parts in the network, for both training and inference, many of the difficult engineering decisions are greatly simplified. Figure 2 provides an overview of the DDNN framework. Each version presented shows how DDNN can scale the inference computation across different physical device configurations. The cloud-based DDNN in (a) can be viewed as the standard DNN running in the cloud as described in the introduction. In this case, sensor input captured on end devices is sent to the cloud in original format, where the all layers of DNN inference is performed.

We can extend this model to include end devices, as shown in (b), by performing a portion of the DNN inference computation on the end device rather than sending the raw input to the cloud. Using an exit point after device inference, we may classify samples, that the local network is confident about, without sending any information to the cloud. For more difficult cases, the intermediate DNN output up to the local exit is sent to the cloud, where further inference is performed using additional NN layers and a final classification decision is made. Note that the intermediate output can be designed to be smaller than the sensor input (i.e., raw image from a video camera), and therefore reduce the network communication required between the end device and the cloud. The details of how communication is considered in the network is discussed in section III-E.

DDNN can also be easily extended to multiple end devices, shown in (c), that work together to make a classification decision. Here, each end device performs local computation as in (b), but their output is aggregated together before the local exit point. We will discuss the design of feature aggregation in detail in section III-B. As before, if the local exit point is not confident about the sample, each end devices sends intermediate output to the cloud, where another round of feature aggregation is performed before making a final classification decision.

DDNN scales vertically as well, by, e.g., introducing an edge layer between the end devices and cloud, shown in (d) and (e). The edge acts similarly to the cloud, by taking output from the end devices, performing aggregation and classification if possible, and forwarding its own intermediate output to the cloud if more processing is needed. In this way, DDNN naturally adjusts the network communication and response time of the system on a per sample basis. Samples that can be correctly classified locally are exiting without any communication to the edge or cloud. Samples that require more feature extraction than can be provided locally are sent to the edge, and eventually the cloud if necessary. Finally, DDNNs can also scale geographically across the edge layer as well, which is shown in (f).

B. DDNN Aggregation Methods

In DDNN configurations with multiple end devices (e.g., (c), (e), and (f) in Figure 2), the output from each end device must be aggregated in order to perform classification. We present several different schemes for aggregating the output. Each aggregation method makes different assumptions about how the device output should be combined and therefore can result in different system accuracy. We present three approaches:

- Max pooling (MP). MP aggregates the input vectors by taking the max of each component. Mathematically, max pooling can be written as
  \[ \hat{v}_j = \max_{1 \leq i \leq n} v_{ij}, \]
  where \( n \) is the number of inputs and \( v_{ij} \) is the \( j \)-th component of the input vector and \( \hat{v}_j \) is the \( j \)-th component of the resulting output vector.

- Average pooling (AP). AP aggregates the input vectors by taking the average of each component. Mathematically, we have
  \[ \hat{v}_j = \frac{1}{n} \sum_{i=1}^{n} v_{ij}, \]
Fig. 2: Overview of the DDNN framework. The vertical lines represent the DNN pipeline, which connect the horizontal bars (layers). (a) is the standard DNN (represented in the cloud), (b) introduces end devices and a local exit point that may classify samples before the cloud, (c) extends (b) by adding multiple end devices which are aggregated together for classification, (d) and (e) extend (b) and (c) by adding edge layers between the cloud and end devices, (f) shows how the edge can also be distributed like the end devices.

where $n$ is the number of inputs and $v_{ij}$ is the $j$-th component of the input vector and $\hat{v}_j$ is the $j$-th component of the resulting output vector. Averaging may reduce noisy input presented in some end devices.

- Concatenation (CC). CC simply concatenates the input vectors together. CC retains all information which is useful for higher layers (e.g., the cloud) that can use the full information to extract higher level features. Note that this expands the dimension of the resulting vector. To map this vector back to the same number of dimensions as input vectors, we add an additional linear layer.

We analyzes these aggregation methods in Section IV-C.

C. DDNN Training

While DDNN is distributed over multiple physical devices at inference time, it can be trained on a single powerful device or in the cloud. One aspect of DDNN that is different from most conventional DNN pipelines is the use of multiple exit points as shown in Figure 2. At training time, the loss from each exit is combined during back-propagation so that the entire network can be jointly trained, and each exit point achieves good accuracy relative to its depth. For this work, we follow joint training as described in GoogleNet and BranchyNet [14, 15].

For the system evaluation discussed in Section IV, we apply DDNNs to a classification task. We use the softmax cross entropy loss function as the optimization objective. We now describe formally how we train DDNNs. Let $y$ be a one-hot ground-truth label vector, $x$ be an input sample and $C$ be the set of all possible labels. For each exit, the softmax cross entropy objective function can be written as

$$L(\hat{y}, y; \theta) = -\frac{1}{|C|} \sum_{c \in C} y_c \log \hat{y}_c,$$

where

$$\hat{y} = \text{softmax}(z) = \frac{\exp(z)}{\sum_{c \in C} \exp(z_c)},$$

and

$$z = f_{exit_n}(x; \theta),$$

where $f_{exit_n}$ is the output of the $n$-th exit branch and $\theta$ represents the parameters of the layers from an entry point to the exit point.
To train the DDNN we form a joint optimization problem as minimizing a weighted sum of the loss functions of each exit:

$$L(\hat{y}, y; \theta) = \sum_{n=1}^{N} w_n L(\hat{y}_{exit_n}, y; \theta),$$

where $N$ is the total number of exit points and $w_n$ is the associated weight of each exit.

### D. DDNN Inference

Inference in DDNN is performed in multiple stages using multiple predefined exit thresholds $T$ (one element $T$ at each exit point) as a measure of confidence in the prediction of the sample. We use a normalized entropy threshold as the confidence criteria (instead of unnormalized entropy as used in [15]) that determines whether to classify (exit) a sample at a particular exit point. The normalized entropy is defined as

$$\eta(x) = -\sum_{i=1}^{n} x_i \log \frac{x_i}{\log n},$$

where $n$ is the number of classes and $x$ is a probability vector. This normalized entropy has values between 0 and 1. Low $\eta$ means that the DDNN is confident about the prediction of the sample; high $\eta$ means it is not confident. At each exit point, $\eta$ is computed and compared against $T$ in order to determine if the sample should exit at that point.

At each exit point in DDNN inference, if the predictor is not confident in the result (i.e. $\eta > T$), the system falls back to a higher exit point in the hierarchy until the last exit is reached which always performs classification. We now provide an example of the inference procedure for a DDNN which has multiple end devices and three exit points (configuration (e)) in Figure 2:

1) Each end device first sends summary information to local aggregator.
2) The local aggregator determines if the combined summary information is sufficient for accurate classification.
3) If so, the sample is classified (exited) and no further information is sent to the edge or cloud.
4) If not, each device sends more detailed information to the edge in order to perform more robust classification.
5) If the edge believes it can correctly classify the sample it does so and no information is sent to the cloud.
6) Otherwise, the edge forwards intermediate computation to the cloud which makes the final classification.

### E. Communication Cost of DDNN Inference

The total communication cost for an end device with the local and cloud aggregator is defined as

$$c = n \times 4 + (1 - l) \times \frac{f \times o}{8}$$

(1)

where $l$ is the percentage of sample exited locally, $n$ is the number of classes (3 in our experiments), $f$ and $o$ are the number of filters and the output size of 1 filter for the last NN layer on the end-device. The 4 corresponds to 4 bytes which are used to represent a floating-point number and the 8 corresponds to 8 bits used to convert the binary output (bits) to bytes. The first term consists of a single floating-point per class, which corresponds to the probability the sample is of that class, is transmitted from the end device to the local aggregator. This step happens regardless of whether the sample is exited locally or at a later exit point. The second term is the communication between end device and cloud which happens $(1 - l)$ fraction of the time, when the sample is exited in the cloud rather than locally.

### F. Accuracy Measures

Throughout the evaluation in Section IV we use different accuracy measures to analyze the various exit points in a DDNN as follows:

- **Local Accuracy** is the accuracy when exiting 100% of samples at the local exit of a DDNN.
- **Edge Accuracy** is the accuracy when exiting 100% of samples at the edge exit of a DDNN.
- **Cloud Accuracy** is the accuracy when exiting 100% of samples at the cloud exit of a DDNN.

**Overall Accuracy** is the accuracy when exiting some percentage of samples at each point in the hierarchy. The samples classified at each exit point are determined by the entropy threshold $T$ for that exit. The impact of $T$ on classification accuracy and communication cost is discussed in Section IV-D.

**Individual Accuracy** is the accuracy of an end device trained separately from DDNN. In the evaluation, individual accuracy for each device is computed by training a single layer NN model on the device which classifies all samples by itself without relying on a local or cloud exit points.

### IV. DDNN System Evaluation

In this section, we evaluate DDNN on a scenario with multiple end devices and demonstrate the following characteristics of the approach:

- DDNNs allow multiple end devices to work collaboratively in order to improve accuracy at both the local and cloud exit points.
- DDNNs seamlessly extend the capability of end devices by offloading difficult samples to the cloud.
- DDNNs have built-in fault tolerance. We illustrate that missing any single end device does not dramatically affect the accuracy of the system.
- DDNNs reduce communication costs compared to traditional system that offloads all input sensor data to the cloud.

We first introduce the DDNN architecture and dataset used in our evaluation.

#### A. Evaluation DDNN Architecture

To accommodate the small memory size of the end devices, we use Binary Neural Network [3] block [2]. We make use of

[A block consists of one or more conventional NN layers]
Fig. 3: Fused binary blocks consisting of one or more standard NN layers. The fused binary fully connected (FC) block is a fully connected layer with \( n \) nodes, batch normalization and binary activation. The fused binary convolution-pool (ConvP) block consists of a convolutional layer with \( f \) filters, a pooling layer, batch normalization and binary activation. The convolution layer has a kernel of size 3x3 with stride 1 and padding 1. The pooling layer has a kernel of size 3x3 with stride 2 and padding 1. These blocks are used as they are presented in [10].

Fig. 4: The DDNN architecture used in the system evaluation. The FC and ConvP blocks in red and blue correspond to layers run on end devices and the cloud respectively. The dashed orange boxes represent the end devices and show which blocks of the DDNN are mapped onto each device. The local aggregator shown in red combines the exit output (a short vector with length equal to the number of classes) from each end device in order to determine if local classification for the given input sample can be performed accurately. If the local exit is not confident (i.e. \( \eta(x) > T \)), the activation output after the last convolutional layer from each end device is sent to the cloud aggregator (shown in blue), which aggregates the input from each device, performs further NN layer processing, and outputs a final classification result. The aggregation of input for multiple end devices is discussed in Section IV-C.

two types of blocks in [10]: the fused binary fully connected (FC) block and fused binary convolution-pool (ConvP) block as shown in Figure [3]. FC blocks consists of a fully connected layer with \( n \) nodes, batch normalization and binary activation. ConvP blocks consists of a convolutional layer with \( f \) filters, a pooling layer and batch normalization and binary activation. A convolution layer has a kernel of size 3x3 with stride 1 and padding 1. A pooling layer has a kernel of size 3x3 with stride 2 and padding 1.

For these experiments, we use version (c) from Figure [2] with six end devices. The system presented can be generalized to a more sophisticated structure which includes an edge layer, as shown in (d), (e) or (f) of Figure [2]. Figure [2] shows a detailed view of each NN layer in the DDNN architecture. In this system, we have six end devices shown in red, a local aggregator, and a cloud aggregator. During training, output from each device is aggregated together at each exit point using one of the aggregation schemes described in Section III-B. We provide detailed analysis on the impact of aggregation schemes at both the local and cloud exit points in Section IV-C. All DDNNs in our experiments are trained with Adam [8] using the following hyper-parameter settings: \( \alpha = 0.001 \), \( \beta_1 = 0.9 \), \( \beta_2 = 0.999 \), and \( \epsilon = 1e-8 \). We train each DDNN for 100 epochs. When training the DDNN, we use equal weights for the local and cloud exit points. We explored heavily weighting both the local exit and the cloud exit, but neither weighting scheme significantly changed the accuracy of the system.

B. Multi-view Multi-camera Dataset

We evaluate the proposed framework on the multi-view multi-camera dataset [11]. This dataset consists of images acquired at the same time from six cameras placed at different locations facing the same general area. For the purpose of our evaluation, we assume that each camera is attached to an end device, which can wirelessly transmit the captured images to a physical endpoint connected to the cloud.

Object bounding box annotations are provided with the dataset. Multiple bounding boxes may exist in a single image, each of which corresponds to a different object in the frame. For each bounding box, we extract an image, and manually synchronize the same object across the multiple devices that the object appears in for the given frame. Examples of the extracted images are shown in Figure [5]. Each row corresponds to a single sample used for classification. We resize each extracted image to a 32x32 RGB pixel image. For each device that a given object does not appear in, we use a blank image and assign a label of -1, meaning that the object is not present in the frame. Labels 0, 1, and 2 correspond to car, bus and person, respectively. Objects that are not present in a frame (i.e., label of -1) are not used during training. We split the dataset into 680 training samples and 171 testing samples. Figure [6] shows the distribution of samples at each device. Due to the imbalanced number of class samples in the dataset, the individual accuracy of each end device differs widely, as shown by the “Individual” curve of Figure [8]. A full description of the training process for the individual NN layers run on end devices and the cloud respectively.

\[ \text{cloud aggregator (shown in blue), which aggregates the input from each device, performs further NN layer processing, and outputs a final classification result. The aggregation of input for multiple end devices is discussed in Section IV-C.} \]
Fig. 5: Example images of three objects (person, bus, car) from the multi-view multi-camera dataset. The six devices (cameras) capture the same object from different orientations. An all grey image denotes that the object is not present in the frame.

Fig. 6: The distribution of class samples for each device in the multi-view multi-camera dataset.

The MP-MP scheme has good classification accuracy for the local aggregator but poor performance in the cloud. The elements in the vectors at the local aggregator correspond to the same features (e.g., the first item is the likelihood that the input corresponds to that class). Therefore, max pooling corresponds to taking the max response for each class over all end devices, which performs well. On the other hand, since the information sent from the end devices to the cloud is the activation output from the filters at each device, which correspond to different visual features in the input from each end device, max pooling these features does not perform well. The CC-CC scheme shows an opposite trend where the local accuracy is poor and the cloud accuracy is high. Concatenating the local information (instead of a pooling scheme), does not enforce any relationship between output for the same class on multiple devices and therefore performs worse. Concatenating the output for the cloud aggregator maintains the most information for NN layer processing in the cloud and therefore performs well. Generally, average pooling appears to perform worse than max pooling. This is because some of the end devices do not have the object present in the frame not have object presented. Average pooling take average of all outputs from end devices; this interferes with the strong outputs from end devices in which the object is present. Based on these results, we use the MP-CC aggregation scheme throughout the paper.

### Table I: Accuracy of different aggregation schemes

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Local Acc. (%)</th>
<th>Cloud Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP-MP</td>
<td>95</td>
<td>91</td>
</tr>
<tr>
<td>MP-CC</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>AP-AP</td>
<td>86</td>
<td>98</td>
</tr>
<tr>
<td>AP-CC</td>
<td>75</td>
<td>96</td>
</tr>
<tr>
<td>CC-CC</td>
<td>85</td>
<td>94</td>
</tr>
<tr>
<td>AP-MP</td>
<td>88</td>
<td>93</td>
</tr>
<tr>
<td>MP-AP</td>
<td>89</td>
<td>97</td>
</tr>
<tr>
<td>CC-MP</td>
<td>77</td>
<td>87</td>
</tr>
<tr>
<td>CC-AP</td>
<td>80</td>
<td>94</td>
</tr>
</tbody>
</table>

The MP-MP scheme has good classification accuracy for the local aggregator but poor performance in the cloud. The elements in the vectors at the local aggregator correspond to the same features (e.g., the first item is the likelihood that the input corresponds to that class). Therefore, max pooling corresponds to taking the max response for each class over all end devices, which performs well. On the other hand, since the information sent from the end devices to the cloud is the activation output from the filters at each device, which correspond to different visual features in the input from each end device, max pooling these features does not perform well. The CC-CC scheme shows an opposite trend where the local accuracy is poor and the cloud accuracy is high. Concatenating the local information (instead of a pooling scheme), does not enforce any relationship between output for the same class on multiple devices and therefore performs worse. Concatenating the output for the cloud aggregator maintains the most information for NN layer processing in the cloud and therefore performs well. Generally, average pooling appear to perform worse than max pooling. This is because some of the end devices do not have the object present in the frame not have object presented. Average pooling take average of all outputs from end devices; this interferes with the strong outputs from end devices in which the object is present. Based on these results, we use the MP-CC aggregation scheme throughout the paper.

### C. Impact of Aggregation Schemes

In order to perform classification on the input from multiple end devices, we must aggregate the information from each end device. We consider three aggregation methods (max pooling, average pooling, and concatenation) outlined in Section III-B at both the local and cloud exit points. The accuracy of different aggregation schemes is shown in Table I. The first two letters represent the local aggregation scheme and the last two letters are the scheme used by the cloud aggregator. For example, MP-CC means the local aggregator uses max-pooling and the cloud uses concatenation. Recall that the input to the local aggregator is a floating-point vector of length equal to the number of classes (corresponding to the output from the final FC block for a single device as shown in Figure 4) and the device output sent to the cloud aggregator is the output from the final ConvP block.

### D. Entropy Threshold

The entropy threshold for an exit point, $T$, corresponds to the level of confidence that is required in order to exit a sample. A threshold value of 0 means that no samples will exit and a value of 1 means that all samples exit at that point. Figure 7 shows the relationship between $T$ at the local aggregator and the overall accuracy of the DDNN. We observe that as more samples are exited at the local exit, the overall
accuracy decreases. This is expected, as the accuracy of the local exit is typically somewhat lower than that of the cloud exit.

We need to set the threshold appropriately to trade off the communication cost, as defined in Section III-E, and accuracy of the system. In this case, we see that setting the threshold to 0.8 results in the best overall accuracy with significantly reduced communication, i.e., 97% accuracy while exiting 60% of samples locally as shown in Table II. We set $T = 0.8$ for the remaining experiments in the system evaluation, unless noted otherwise.

TABLE II: Effects of different exit threshold ($T$) settings for the local exit. $T = 0.8$ is used in the remaining experiments.

<table>
<thead>
<tr>
<th>$T$</th>
<th>Local Exit (%)</th>
<th>Overall Acc. (%)</th>
<th>Comm. (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.00</td>
<td>96</td>
<td>140</td>
</tr>
<tr>
<td>0.3</td>
<td>0.58</td>
<td>96</td>
<td>139</td>
</tr>
<tr>
<td>0.5</td>
<td>1.75</td>
<td>96</td>
<td>138</td>
</tr>
<tr>
<td>0.6</td>
<td>2.92</td>
<td>96</td>
<td>136</td>
</tr>
<tr>
<td>0.7</td>
<td>22.81</td>
<td>96</td>
<td>111</td>
</tr>
<tr>
<td>0.8</td>
<td>60.82</td>
<td>97</td>
<td>62</td>
</tr>
<tr>
<td>0.9</td>
<td>83.04</td>
<td>96</td>
<td>34</td>
</tr>
<tr>
<td>1.0</td>
<td>100.00</td>
<td>92</td>
<td>12</td>
</tr>
</tbody>
</table>

E. Impact of Scaling across End Devices

In order to scale DDNNs across multiple end devices, we simply distribute the lower sections of Figure 4, shown in red, over the corresponding devices, outlined in orange. Figure 8 shows how the accuracy of the system improves as additional end devices (attached to the input cameras) are added. The devices are added in order sorted by their individual accuracy from worst to best (e.g., the device with the lowest accuracy first and the device with the highest accuracy last).

The first observation is the large variation in the individual accuracy of the end devices, as noted earlier. Due to the nature of the dataset, some devices are naturally better positioned and generally have more clear observations of the objects. Looking at the viewpoints of each camera in Figure 5, we see that the selected examples for Device 6 have clear frontal views of each object. This viewpoint gives Device 6 the highest individual accuracy at over 70%. By comparison, Device 2 has the lowest individual accuracy at under 40%.

The “Local” and “Cloud” curves show the accuracy of the system at each exit point when all samples are exited at that point. We observe that the cloud exit point out performs the local exit point at all numbers of end devices. The gap is widest when there are fewer devices, which suggests that the additional NN layers in the cloud significantly improve the final classification result when the problem is more difficult due to limited labeled training data for an end device. Once all 6 end devices are added, both the local and cloud aggregators have high accuracy. The “Overall” curve represents the overall accuracy of the system when the threshold for the local exit point is set to 0.8. We see that this curve is roughly equivalent to exiting all samples at the cloud (but at a much reduced communication cost as 57% of samples are exited locally). Generally, these results show that by combining multiple viewpoints we can increase the classification accuracy at both the local and cloud level by a substantial margin when compared to the individual accuracy of any device. The resulting accuracy of the DDNN system is superior to any individual device accuracy by over 20%.

F. Impact of Cloud Offloading on Accuracy Improvements

DDNNs improve the overall accuracy of the system by offloading difficult samples to the cloud, which perform further
Fig. 9: Accuracy and communication cost for increasingly larger end device memory sizes that accommodate additional filters. We notice that cloud offloading leads to improved accuracy.

NN layer processing and final classification. Figure 9 shows the accuracy and communication costs of DDNN as the number of filters on the end devices increases. In this experiment, we configure the local exit threshold $T$ such that around 75\% of samples are exited locally and around 25\% of samples are offloaded to the cloud. We see that DDNNs achieve around 5\% improvement in accuracy compared to using just the local aggregator. This demonstrates the need for offloading to the cloud even when larger models (more filters) with improved local accuracy are used on the end devices.

G. Fault Tolerance of DDNNs

A key design feature of distributed systems is fault tolerance. Fault tolerance implies that the system still works well even when some parts are broken. In order to test the fault tolerance of DDNN, we simulate end device failures and look at the resulting accuracy of the system. Figure 10 shows the accuracy of the system under the presence of individual device failures. Regardless of the device that is missing, the system still achieves over a 95\% overall classification accuracy. Specifically, even when the device with the highest individual accuracy has failed, which is Device 6, the overall accuracy is reduced by only 3\%. This suggests that for this dataset, the automatic fault tolerance provided by DDNN makes the system reliable even in the presence of device failure.

H. Reducing Communication Costs

DDNNs significantly reduces the communication cost of inference compared to the standard method of offloading raw sensor input to the cloud. Sending a 32x32 RGB pixel image (the input size of our dataset) to the cloud costs 3072. By comparison, as shown in Table II, the largest DDNN model used in our evaluation section requires only 140 bytes of communication per sample on average (an over 20x reduction in communication costs). This communication reduction for an end device results from transmitting class-label related intermediate results to the local aggregator for all samples and binarized communication with the cloud when additional NN layer processing is required for correct classification.

V. DDNN Provision for Horizontal and Vertical Scaling

The evaluation in the previous section shows that DDNN is able to achieve high overall accuracy through provisioning the network to scale both horizontally, across end devices, and vertically, over the network hierarchy. Specifically, we show that DDNN scales vertically, by exiting easier input samples locally and offloading difficult samples to the cloud, while maintaining a small memory footprint on the end devices and achieving a high overall accuracy. This result is not obvious, as we need sufficiently good feature representations from the lower parts of the DNN (running on the end devices with limited resources) in order for the upper parts of the network (running in the cloud) to achieve high accuracy. Therefore, we show in a positive way that the proposed method of jointly training a single DNN with multiple exit points at each part of the distributed hierarchy allows us to meet this goal. That is, DDNN optimizes the lower parts of the DNN to create a sufficiently good feature representations to support both samples exited locally and those processed further in the cloud.

To meet the goal of horizontal scaling, we provide a principled way of jointly training a DNN with inputs from multiple devices through feature pooling via local and cloud aggregators and demonstrate that by aggregating features from each device we can dramatically improve the accuracy of the system both at the local and cloud level. Filters on each device
are specifically tuned to process the geographically unique inputs and work together toward the same overall objective leading to high overall accuracy. Additionally, we show that DDNN provides built-in fault tolerance across the end devices and is still able to achieve high accuracy in the absence of any single device.

VI. CONCLUSION

In this paper, we propose a novel distributed deep neural network architecture (DDNN) that is distributed across computing hierarchies, consisting of the cloud, the edge and end devices. We demonstrate for a simple, yet challenging multi-view, multi-camera dataset that DDNN scales vertically from a few NN layers on end devices to many NN layers in the cloud and scales horizontally across multiple end devices. This result suggests that with our DDNN framework, a single DNN properly trained can be mapped onto a distributed computing hierarchy to meet the accuracy requirements of a target application while gaining multiple benefits associated with distributed hierarchies such as fault tolerance.

DDNNs also reduce the required communication compared to a standard cloud offloading approach by exiting many samples at the local aggregator and sending a compact binary feature representation to the cloud when additional processing is required. For our evaluation dataset, the communication cost of DDNN is reduced by a factor of over 20x compared to offloading raw sensor input to a DNN in the cloud which performs all of the inference computation.

For future work, we are interested in investigating the performance of DDNNs on more applications with multiple types of input modalities [2] and more end devices. Additionally, we will explore mixed precisions schemes where the end devices use binary NN layers and the cloud uses floating-point NN layers. Currently, all layers in DDNN are binary. While this is a requirement for end devices due to the limited space on device, it is not necessary in the cloud. The DDNN codebase is open source and can be found here: https://gitlab.com/htkung/ddnn

REFERENCES