JointDNN: An Efficient Training and Inference Engine for Intelligent Mobile Cloud Computing Services

Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram, *Fellow, IEEE*

Abstract—Deep learning models are being deployed in many mobile intelligent applications. End-side services, such as intelligent personal assistants, autonomous cars, and smart home services often employ either simple local models on the mobile or complex remote models on the cloud. However, recent studies have shown that partitioning the DNN computations between the mobile and cloud can increase the latency and energy efficiencies. In this paper, we propose an efficient, adaptive, and practical engine, JointDNN, for collaborative computation between a mobile device and cloud for DNNs in both inference and training phase. JointDNN not only provides an energy and performance efficient method of querying DNNs for the mobile side but also benefits the cloud server by reducing the amount of its workload and communications compared to the cloud-only approach. Given the DNN architecture, we investigate the efficiency of processing some layers on the mobile device and some layers on the cloud server. We provide optimization formulations at layer granularity for forward- and backward-propagations in DNNs, which can adapt to mobile battery limitations and cloud server load constraints and quality of service. JointDNN achieves up to 18 and 32 times reductions on the latency and mobile energy consumption of querying DNNs compared to the status-quo approaches, respectively.

Index Terms—Deep neural networks, intelligent services, mobile computing, cloud computing

1 Introduction

DNN architectures are promising solutions in achieving remarkable results in a wide range of machine learning applications, including, but not limited to computer vision, speech recognition, language modeling, and autonomous cars. Currently, there is a major growing trend in introducing more advanced DNN architectures and employing them in end-user applications. The considerable improvements in DNNs are usually achieved by increasing computational complexity which requires more resources for both training and inference [1]. Recent research directions to make this progress sustainable are: development of Graphical Processing Units (GPUs) as the vital hardware component of both servers and mobile devices [2], design of efficient algorithms for large-scale distributed training [3] and efficient inference [4], compression and approximation of models [5], and most recently introducing collaborative computation of cloud and fog as known as dew computing [6].

Deployment of cloud servers for computation and storage is becoming extensively favorable due to technical advancements and improved accessibility. Scalability, low cost, and satisfactory Quality of Service (QoS) made offloading to cloud a typical choice for computing-intensive tasks. On the other side, mobile-device are being equipped with more powerful general-purpose CPUs and GPUs. Very recently there is a new trend in hardware companies to design dedicated chips to better tackle machine-learning tasks. For example, Apple's A11 Bionic chip [7] used in iPhone X uses a neural engine in its GPU to speed up the DNN queries of applications such as face identification and facial motion capture [8].

In the status-quo approaches, there are two methods for DNN inference: mobile-only and cloud-only. In simple models, a mobile device is sufficient for performing all the computations. In the case of complex models, the raw input data (image, video stream, voice, etc.) is uploaded to and then the required computations are performed on the cloud server. The results of the task are later downloaded to the device. The effects of raw input and feature compression are studied in [9] and [10].

Despite the recent improvements of the mobile devices mentioned earlier, the computational power of mobile devices is still significantly weaker than the cloud ones. Therefore, the mobile-only approach can cause large inference latency and failure in meeting QoS. Moreover, embedded devices undergo major energy consumption constraints due to battery limits. On the other hand, cloud-only suffers communication overhead for uploading the raw data and downloading the outputs. Moreover, slowdowns caused by service congestion, subscription costs, and network dependency should be considered as downsides of this approach [11].

The superiority and persistent improvement of DNNs depend heavily on providing a huge amount of training data. Typically, this data is collected from different resources and later fed into a network for training. The final model

• *The authors are with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089-2562 USA. E-mail: {eshratif, abri442, pedram}@usc.edu.*

Manuscript received 6 Sept. 2018; revised 9 Oct. 2019; accepted 13 Oct. 2019. Date of publication 16 Oct. 2019; date of current version 7 Jan. 2021. (Corresponding author: Amir Erfan Eshratifar.) Digital Object Identifier no. 10.1109/TMC.2019.2947893

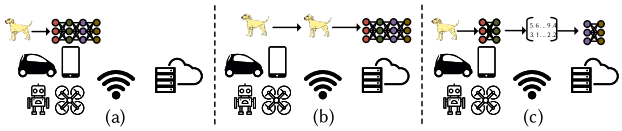

Fig. 1. Different computation partitioning methods. (a) Mobile only: computation is completely done on the mobile device. (b) Cloud only: raw input data is sent to the cloud server, computations is done on the cloud server and results are sent back to the mobile device. (c) JointDNN: DNN architecture is partitioned at the granularity of layers, each layer can be computed either on cloud or mobile.

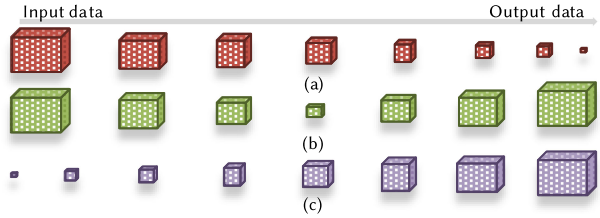

Fig. 2. Typical layer size in (a) Discriminative, (b) Autoencoder, and (c) Generative models.

can then be delivered to different devices for inference functions. However, there is a trend of applications requiring adaptive learning in online environments, such as self-driving cars and security drones [12], [13]. Model parameters in these smart devices are constantly being changed based on their continuous interaction with their environment. The complexity of these architectures with an increasing number of parameters and current cloud-only methods for DNN training implies a constant communication cost and the burden of increased energy consumption for mobile devices. The main difference of collaborative training and cloud-only training is that the data transferred in the cloud-only approach is the input data and model parameters but in the collaborative approach, it is layer(s)'s output and a portion of model parameters. Therefore, the amount of data communicated can be potentially decreased [14].

Automatic partitioning of computationally extensive tasks over the cloud for optimization of performance and energy consumption has been already well-studied [15]. Most recently, scalable distributed hierarchy structures between the end-user device, edge, and cloud have been suggested [16] which are specialized for DNN applications. However, exploiting the layer granularity of DNN architectures for runtime partitioning has not been studied thoroughly yet.

In this work, we are investigating the inference and training of DNNs in a *joint* platform of mobile and cloud as an alternative to the current single-platform methods as illustrated in Fig. 1. Considering DNN architectures as an ordered sequence of layers, and the possibility of computation of every layer either on mobile or cloud, we can model the DNN structure as a Directed Acyclic Graph (DAG). The parameters of our real-time adaptive model are dependent on the following factors: mobile/cloud hardware and software resources, battery capacity, network specifications, and QoS. Based on this modeling, we show that the problem of finding the optimal computation schedule for different scenarios, i.e., best performance or energy consumption, can be reduced to the polynomial-time shortest path problem.

To present realistic results, we made experiments with fair representative hardware of mobile device and cloud. To model the communication costs between platforms, we used various mobile network technologies and the most recent reports on their specifications in the U.S.

DNN architectures can be categorized based on functionality. These differences enforce specific type and order of layers in architecture, directly affecting the partitioning result in the collaborative method. For discriminative models, used in recognition applications, the layer size gradually decreases going from input toward output as shown in Fig. 2. This sequence suggests the computation of the first few layers on the mobile device to avoid excessive communication cost of uploading large raw input data. On the other hand, the growth of the layer output size from input to output in generative models which are used for synthesizing new data, implies the possibility of uploading a small input vector to the cloud and later downloading one of the last layers and performing the rest of computations on the mobile device for better efficiency. Interesting mobile applications like image-to-image translation are implemented with autoencoder architectures whose middle layers sizes are smaller compared to their input and output. Consequently, to avoid huge communication costs, we expect the first and last layers to be computed on the mobile device in our collaborative approach. We examined eight well-known DNN benchmarks selected from these categories to illustrate their differences in the collaborative computation approach.

As we will see in Section 4, the communication between the mobile and cloud is the main bottleneck for both performance and energy in the collaborative approach. We investigated the specific characteristics of CNN layer outputs and introduced a loss-less compression approach to reduce the communication costs while preserving the model accuracy.

State-of-the-art work for collaborative computation of DNNs [14] only considers one offloading point, assigning computation of its previous layers and next layers on the mobile and cloud platforms, respectively. We show that this approach is non-generic and fails to be optimal, and introduced a new method granting the possibility of computation on either platform for each layer independent of other layers. Our evaluations show that JointDNN significantly improves the latency and energy up to $3\times$ and $7\times$ respectively compared to the status-quo single platform approaches without any compression. The main contributions of this paper can be listed as:

- Introducing a new approach for the collaborative computation of DNNs between the mobile and cloud
- Formulating the problem of optimal computation scheduling of DNNs at layer granularity in the mobile cloud computing environment as the shortest path problem and integer linear programming (ILP)
- Examining the effect of compression on the outputs of DNN layers to improve communication costs
- Demonstrating the significant improvements in performance, mobile energy consumption, and cloud workload achieved by using *JointDNN*

2 Problem Definition and Modeling

In this section, we explain the general architecture of DNN layers and our profiling method. Moreover, we elaborate on

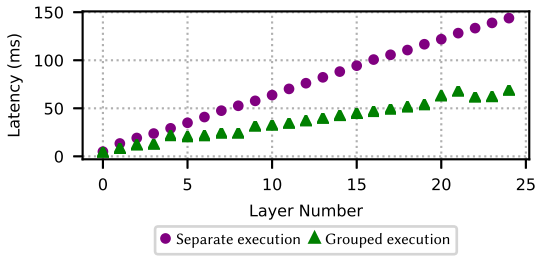

Fig. 3. Latency of grouped and separated execution of convolution operator.

how cost optimization can be reduced to the shortest path problem by introducing the JointDNN graph model. Finally, we show how the constrained problem is formulated by setting up ILP.

2.1 Energy and Latency Profiling

There are three methods in measuring the latency and energy consumption of each layer in neural networks [17]:

Statistical Modeling. In this method, a regression model over the configurable parameters of operators (e.g., filter size in the convolution) can be used to estimate the associated latency and energy. This method is prone to large errors because of the inter-layer optimizations performed by DNN software packages. Therefore, it is necessary to consider the execution of several consecutive operators grouped during profiling. Many of these software packages are proprietary, making access to inter-layer optimization techniques impossible.

In order to illustrate this issue, we designed two experiments with 25 consecutive convolutions on NVIDIA Pascal™ GPU using cuDNN® library [18]. In the first experiment, we measure the latency of each convolution operator separately and set the total latency as the sum of them. In the second experiment, we execute the grouped convolutions in a single kernel together and measure the total latency. All parameters are located on the GPU's memory in both experiments, avoiding any data transfer from the main memory to make sure results are exactly representing the actual computation latency.

As we see in Fig. 3, there is a large error gap between separated and grouped execution experiments which grows as the number of convolutions is increased. This observation confirms that we need to profile grouped operators to have more accurate estimations. Considering the various consecutive combination of operators and different input sizes, this method requires a very large number of measurements, not to mention the need for a complex regression model.

Analytical Modeling. To derive analytical formulations for estimating the latency and energy consumption, it is required to obtain the exact hardware and software specifications. However, the state-of-the-art in latency modeling of DNNs [19] fails to estimate layer-level delay within an acceptable error bound, for instance, underestimating the latency of a fully connected layer with 4,096 neurons by around 900 percent. Industrial developers do not reveal the detailed hardware architecture specifications and the proprietary parallel computing architectures such as CUDA®, therefore, the analytical approach could be quite challenging [20].

Application-Specific Profiling. In this method, the DNN architecture of the application being used is profiled in run-time. The number of applications in a mobile device using neural networks is generally limited. In conclusion, this method is more feasible, promising higher accuracy estimations. We have chosen this method for the estimation of energies and latencies in the experiments of this paper.

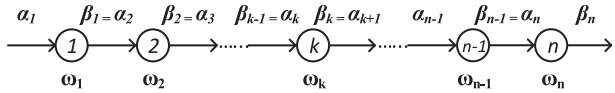

Fig. 4. Computation model in linear topology.

2.2 JointDNN Graph Model

First, we assume that a DNN is presented by a sequence of distinct layers with a linear topology as depicted in Fig. 4. Layers are executed sequentially, with output data generated by one layer feeds into the input of the next one. We denote the input and output data sizes of kth layer as α_k and β_k, respectively. Denoting the latency (energy) of layer k as ω_k, where $k = 1, 2, \ldots, n$, the total latency (energy) of querying the DNN is $\sum_{k=1}^{n} \omega_k$.

The mobile cloud computing optimal scheduling problem can be reduced to the shortest path problem, from node S to F, in the graph of Fig. 5. *Mobile Execution* cost of the kth layer ($C(ME_k)$) is the cost of executing the kth layer in the mobile while the cloud server is idle. *Cloud Execution* cost of the kth layer ($C(CE_k)$) is the executing cost of the kth layer in the cloud server while the mobile is idle. *Uploading the Input Data* cost of the kth layer is the cost of uploading output data of the (k-1)th layer to the cloud server (UID_k). *Downloading the Input Data* cost of the kth layer is the cost of downloading output data of the (k-1)th layer to the mobile (DOD_k). The costs can refer to either latency or energy. However, as we showed in Section 2.1, the assumption of linear topology in DNNs is not true and we need to consider all the consecutive grouping of the layers in the network. This fact suggests the replacement of linear topology by a tournament graph as depicted in Fig. 6. We define the parameters of this new graph, *JointDNN graph model*, in Table 1.

In this graph, node $C_{i:j}$ represents that the layers i to j are computed on the cloud server, while node $M_{i:j}$ represents that the layers i to j are computed on the mobile device. An edge between two adjacent nodes in JointDNN graph model is associated with four possible cases: 1) A transition from the mobile to the mobile, which only includes the mobile computation cost ($ME_{i,j}$) 2) A transition from the cloud to the cloud, which only includes the cloud computation cost ($CE_{i,j}$) 3) A transition from the mobile to the cloud, which includes the mobile computation cost and uploading cost of

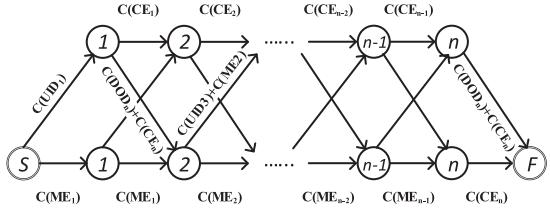

Fig. 5. Graph representation of mobile cloud computing optimal scheduling problem for linear topology.

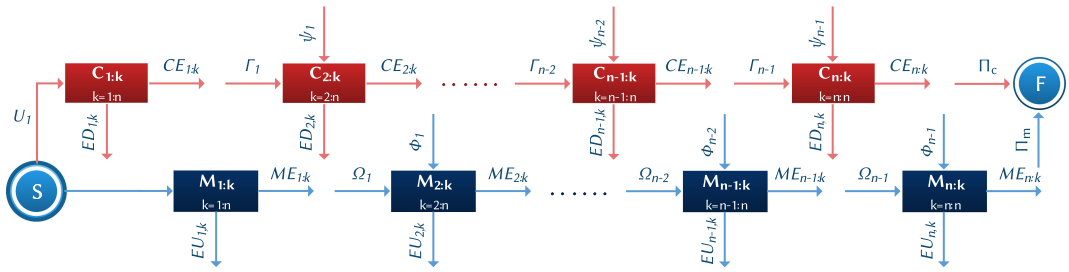

Fig. 6. JointDNN graph model. The shortest path from S to F determines the schedule of executing the layers on mobile or cloud.

the inputs of the next node ($EU_{i,j} = ME_{i,j} + UID_{j+1}$) 4) A transition from the cloud to the mobile, which includes the cloud computation cost and downloading cost of the inputs of the next node ($ED_{i,j} = CE_{i,j} + DOD_{j+1}$). Under this formulation, we can transform the computation scheduling problem to finding the shortest path from S to F. Residual networks are a class of powerful and easy-to-train architectures of DNNs [21]. In residual networks, as depicted in Fig. 7a, the output of one layer is fed into another layer with a distance of at least two. Thus, we need to keep track of the source layer (node 2 in Fig. 7) to know that this layer is computed on the mobile or the cloud. Our standard graph model has a memory of one which is the very previous layer. We provide a method to transform the computation graph of this type of network to our standard model, JointDNN graph.

In this regard, we add two additional chains of size $k-1$, where k is the number of nodes in the residual block (3 in Fig. 7). One chain represents the case of computing layer 2 on the mobile and the other one represents the case of computing layer 2 on the cloud. In Fig. 7, we have only shown the weights that need to be modified, where D_2 and U_2 are the cost of downloading and uploading the output of layer 2, respectively.

By solving the shortest path problem in the JointDNN graph model, we can obtain the optimal scheduling of inference in DNNs. The online training consists of one inference and one back-propagation step. The total number of layers is noted by N consistently throughout this paper so there are $2N$ layers for modeling training, where the second N layers are the mirrored version of the first N layers, and their associated operations are the gradients of the error function concerning the DNN's weights. The main difference between the mobile cloud computing graph of inference and online training is the need for updating the model by downloading the new weights from the cloud. We assume that the cloud server performs the whole back-propagation step separately, even if it is scheduled to be done on the mobile, therefore, there is no need for the mobile device to upload the weights that are updated by itself to save mobile energy consumption. The modification in the JointDNN graph model is adding the costs of downloading weights of the layers that are updated in the cloud to $ED_{i,j}$. The shortest path problem can be solved in polynomial time efficiently.

However, the problem of the shortest path subjected to constraints is NP-Complete [22]. For instance, assuming our standard graph is constructed for energy and we need to find the shortest path subject to the constraint of the total latency of that path is less than a time deadline (QoS). However, there is an approximation solution to this problem, "LARAC" algorithm [23], the nature of our application does not require to solve this optimization problem frequently, therefore, we aim to obtain the optimal solution. We can constitute a small look-up table of optimization results for a different set of parameters (e.g., network bandwidth, cloud server load, etc.). We provide the ILP formulations of DNN partitioning in the following sections.

TABLE 1
Parameter Definition of Graph Model

Param.	Description of Cost
$CE_{i:j}$	Executing layers i to j on the cloud
$ME_{i:j}$	Executing layers i to j on the mobile
$ED_{i,j}$	$CE_{i:j} + DOD_j$
$EU_{i,j}$	$ME_{i:j} + UID_j$
ϕ_k	All the following edges: $\forall i = 1 : k-1$ $ED_{i,k-1}$
Ω_k	All the following edges: $\forall i = 1 : k-1$ $ME_{i,k-1}$
Ψ_k	All the following edges: $\forall i = 1 : k-1$ $EU_{i,k-1}$
Γ_k	All the following edges: $\forall i = 1 : k-1$ $CE_{i,k-1}$
Π_m	All the following edges: $\forall i = 1 : n$ $ME_{i,n}$
Π_c	All the following edges: $\forall i = 1 : n$ $ED_{i,n}$
U_1	Uploading the input of the first layer

2.3 ILP Setup

2.3.1 Performance Efficient Computation Offloading ILP Setup for Inference

We formulated the scheduling of inference in DNNs as an ILP with tractable number of variables. In our method, first we profile the delay and energy consumption of consecutive layers of size $m \in \{1, 2, \ldots, N\}$. Thus, we will have

$$N + (N-1) + \ldots + 1 = N(N+1)/2, \quad (1)$$

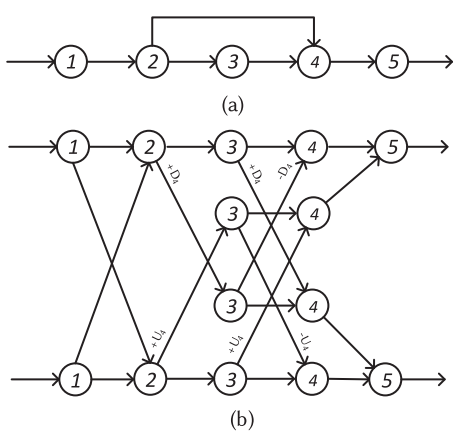

Fig. 7. (a) A residual building block in DNNs. (b) Transformation of a residual building block to be able to be used in JointDNN's shortest path based scheduler.

number of different profiling values for delay and energy. Considering layer i to layer j to be computed either on the mobile device or cloud server, we assign two binary variables $m_{i,j}$ and $c_{i,j}$, respectively. Download and upload communication delays needs to be added to the execution time, when switching from/to cloud to/from mobile, respectively.

$$T_{computation} = \sum_{i=1}^{n}\sum_{j=i}^{n}(m_{i,j}.T_{mobile_{L_{i,j}}} + c_{i,j}.T_{cloud_{L_{i,j}}}) \tag{2}$$

$$\begin{aligned} T_{communication} = &\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j+1}^{n} m_{i,j}.c_{j+1,k}.T_{upload_{L_j}} \\ &+ \sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j+1}^{n} c_{i,j}.m_{j+1,k}.T_{download_{L_j}} \\ &+ \sum_{i=1}^{n} c_{1,i}.T_{upload_{L_i}} \\ &+ \sum_{i=1}^{n} c_{i,n}.T_{download_{L_n}} \end{aligned} \tag{3}$$

$$T_{total} = T_{computation} + T_{communication}. \tag{4}$$

$T_{mobile_{L_{i,j}}}$ and $T_{cloud_{L_{i,j}}}$ represent the execution time of the ith layer to the jth layer on the mobile and cloud, respectively. $T_{download_{L_i}}$ and $T_{upload_{L_i}}$ represent the latency of downloading and uploading the output of the ith layer, respectively. Considering each set of the consecutive layers, whenever $m_{i,j}$ and one of $\{c_{j+1,k}\}_{k=j+1:n}$ are equal to one, the output of the jth layer is uploaded to the cloud. The same argument applies to downloading. We also note that the last two terms in Eq. (3) represent the condition by which the last layer is computed on the cloud and we need to download the output to the mobile device, and the first layer is computed on the cloud and we need to upload the input to the cloud, respectively. To support for residual architectures, we need to add a pair of download and upload terms similar to the first two terms in Eq. (3) for the starting and ending layers of each residual block. In order to guarantee that all layers are computed exactly once, we need to add the following set of constraints:

$$\forall m \in 1:n: \sum_{i=1}^{m}\sum_{j=m}^{n}(m_{i,j} + c_{i,j}) = 1. \tag{5}$$

Because of the non-linearity of multiplication, an additional step is needed to transform Eq. (3) to the standard form of ILP. We define two sets of new variables

$$\begin{aligned} u_{i,j} &= m_{i,j}.\sum_{k=j+1}^{n} c_{j+1,k} \\ d_{i,j} &= c_{i,j}.\sum_{k=j+1}^{n} m_{j+1,k}, \end{aligned} \tag{6}$$

with the following constraints:

$$\begin{aligned} &u_{i,j} \le m_{i,j} \\ &u_{i,j} \le \sum_{k=j+1}^{n} c_{j+1,k} \\ &m_{i,j} + \sum_{k=j+1}^{n} c_{j+1,k} - u_{i,j} \le 1 \\ &d_{i,j} \le c_{i,j} \\ &d_{i,j} \le \sum_{k=j+1}^{n} m_{j+1,k} \\ &c_{i,j} + \sum_{k=j+1}^{n} m_{j+1,k} - d_{i,j} \le 1. \end{aligned} \tag{7}$$

The first two constraints ensure that $u_{i,j}$ will be zero if either $m_{i,j}$ or $\sum_{l=j+1}^{n} c_{j+1,l}$ are zero. The third inequality guarantees that $u_{i,j}$ will take value one if both binary variables, $m_{i,j}$ and $\sum_{l=j+1}^{n} c_{j+1,l}$, are set to one. The same reasoning works for $d_{i,j}$. In summary, the total number of variables in our ILP formulation will be $4N(N+1)/2$, where N is total number of layers in the network.

2.3.2 Energy Efficient Computation Offloading ILP Setup for Inference

Because of the nature of the application, we only care about the energy consumption on the mobile side. We formulate ILP as follows:

$$E_{computation} = \sum_{i=1}^{n}\sum_{j=i}^{n} m_{i,j}.E_{mobile_{L_{i,j}}} \tag{8}$$

$$\begin{aligned} E_{communication} = &\sum_{i=2}^{n}\sum_{j=i}^{n} m_{i,j}.E_{download_{L_i}} \\ &+ \sum_{i=1}^{n}\sum_{j=i}^{n-1} m_{i,j}.E_{upload_{L_j}} \\ &+ \left(\sum_{i=1}^{n}(1 - m_{1,i}) - (n-1)\right).E_{upload_{L_1}} \\ &+ \left(\sum_{i=1}^{n}(1 - m_{i,n}) - (n-1)\right).E_{download_{L_n}} \end{aligned} \tag{9}$$

$$E_{total} = E_{computation} + E_{communication}, \tag{10}$$

$E_{mobile_{L_{i,j}}}$ and $E_{cloud_{L_{i,j}}}$ represent the amount of energy required to compute the ith layer to the jth layer on the mobile and cloud, respectively. $E_{download_{L_i}}$ and $E_{upload_{L_i}}$ represent the energy required to download and upload the output of ith layer, respectively. Similar to performance efficient ILP constraints, each layer should be executed exactly once

$$\forall m \in 1:n: \sum_{i=1}^{m}\sum_{j=m}^{n} m_{i,j} \le 1. \tag{11}$$

The ILP problem can be solved for different set of parameters (e.g., different uplink and download speeds), and then the scheduling results can be stored as a look-up table in the mobile device. Moreover because the number of variables in this setup is tractable solving ILP is quick. For instance, solving ILP for AlexNet takes around 0.045 seconds on Intel (R) Core(TM) i7-3770 CPU with MATLAB®'s intlinprog() function using primal simplex algorithm.

2.3.3 Performance Efficient Computation Offloading ILP Setup for Training

The ILP formulation of online training phase is very similar to that of inference. In online training we have $2N$ layers instead of N obtained by mirroring the DNN, where the second N layers are backward propagation. Moreover, we need to download the weights that are updated in the cloud to the mobile. We assume that the cloud server always has the most updated version of the weights and does not require the mobile device to upload the updated weights. The following terms need to be added for the ILP setup of training

$$T_{computation} = \sum_{i=1}^{2n}\sum_{j=i}^{2n}(m_{i,j}.T_{mobile_{L_{i,j}}} + c_{i,j}.T_{cloud_{L_{i,j}}}) \tag{12}$$

$$\begin{aligned} T_{communication} = &\sum_{i=1}^{2n}\sum_{j=i}^{2n}\sum_{k=j+1}^{2n} m_{i,j}.c_{j+1,k}.T_{upload_{L_j}} \\ &+ \sum_{i=1}^{2n}\sum_{j=i}^{2n}\sum_{k=j+1}^{2n} c_{i,j}.m_{j+1,k}.T_{download_{L_j}} \\ &+ \sum_{i=1}^{n} c_{1,i}.T_{upload_{L_i}} \\ &+ \sum_{i=n+1}^{2n}\sum_{j=i}^{2n} c_{i,j}.T_{download_{W_i}} \end{aligned} \tag{13}$$

$$T_{total} = T_{computation} + T_{communication}. \tag{14}$$

2.3.4 Energy Efficient Computation Offloading ILP Setup for Training

$$E_{computation} = \sum_{i=1}^{2n}\sum_{j=i}^{2n} m_{i,j}.E_{mobile_{L_{i,j}}} \tag{15}$$

$$\begin{aligned} E_{communication} = &\sum_{i=2}^{2n}\sum_{j=i}^{2n} m_{i,j}.E_{download_{L_i}} \\ &+ \sum_{i=1}^{2n}\sum_{j=i}^{2n-1} m_{i,j}.E_{upload_{L_j}} \\ &+ \left(\sum_{i=1}^{2n}(1-m_{1,i}) - (2n-1)\right).E_{upload_{L_1}} \\ &+ \left(\sum_{i=n+1}^{2n}\sum_{j=i}^{2n}(1-m_{i,j}) - (n-1)\right).E_{download_{W_i}} \end{aligned} \tag{16}$$

$$E_{total} = E_{computation} + E_{communication}. \tag{17}$$

Algorithm 1. JointDNN Engine Optimal Scheduling of DNNs

```
function JointDNN (N, L_i, D_i, NB, NP);
  Input: 1: N: number of layers in the DNN
         2: L_i|i = 1 : N: layers in the DNN
         3: D_i|i = 1 : N: data size at each layer
         4: NB: mobile network bandwidth
         5: NP: mobile network uplink and downlink power consumption
  Output: Optimal schedule of DNN
for i = 0; i < N; i = i + 1 do
  for j = 0; j < N; j = j + 1 do
    Latency_{i,j}, Energy_{i,j} = ProfileGroupedLayers(i, j);
  end
end
G,S,F = ConstructShortestPathGraph(N, L_i, D_i, NB, NP) //S and F are start and finish nodes and G is the JointDNN graph model
if no constraints then
  schedule = ShortestPath(G,S,F)
else
  if Battery Limited Constraint then
    E_comm + E_comp ≤ E_ubound
    schedule = PerformanceEfficientILP(N, L_i, D_i, NB, NP)
  end
  if Cloud Server Contraint then
    Σ_{i=1}^{n} Σ_{j=i}^{n} c_{i,j}.T_{cloud_{L_{i,j}}} ≤ T_ubound
    schedule = PerformanceEfficientILP(N, L_i, D_i, NB, NP)
  end
  if QoS then
    T_comm + T_comp ≤ T_QoS
    schedule = EnergyEfficientILP(N, L_i, D_i, NB, NP)
  end
  ;
end
return schedule;
```

2.3.5 Scenarios

There can be different optimization scenarios defined for ILP as listed below:

- *Performance efficient computation:* In this case, it is sufficient to solve the ILP formulation for performance efficient computation offloading.
- *Energy efficient computation:* In this case, it is sufficient to solve the ILP formulation for energy efficient computation offloading.
- *Battery budget limitation:* In this case, based on the available battery, the operating system can decide to dedicate a specific amount of energy consumption to each application. By adding the following constraint to the performance efficient ILP formulation, our framework would adapt to battery limitations

$$E_{computation} + E_{communication} \leq E_{ubound}, \tag{18}$$

- *Cloud limited resources:* In the presence of cloud server congestion or limitations on user's subscription, we can apply execution time constraints to each application to alleviate the server load

TABLE 2
Benchmark Specifications

Type	Model	Layers
Discriminative	AlexNet	21
	OverFeat	14
	Deep Speech	10
	ResNet	70
	VGG16	37
	NiN	29
Generative	Chair	10
Autoencoder	Pix2Pix	32

$$\sum_{i=1}^{n}\sum_{j=i}^{n} c_{i,j}.T_{cloud_{L_{i,j}}} \leq T_{ubound}, \tag{19}$$

- *QoS:* In this scenario, we minimize the required energy consumption while meeting a specified deadline

$$\begin{aligned} &min\{E_{computation} + E_{communication}\} \\ &T_{computation} + T_{communication} \leq T_{QoS}. \end{aligned} \tag{20}$$

This constraint could be applied to both energy and performance efficient ILP formulations.

3 Evaluation

3.1 Deep Architecture Benchmarks

Since the architecture of neural networks depends on the type of application, we have chosen three common application types of DNNs as shown in Table 2:

1) *Discriminative neural networks* are a class of models in machine learning for modeling the conditional probability distribution $P(y|x)$. This class generally is used in classification and regression tasks. AlexNet [24], OverFeat [25], VGG16 [26], Deep Speech [27], ResNet [21], and NiN [28] are well-known discriminative models we use as benchmarks in this experiment. Except for Deep Speech, used for speech recognition, all other benchmarks are used in image classification tasks.
2) *Generative neural networks* model the joint probability distribution $P(x, y)$, allowing generation of new samples. These networks have applications in Computer Vision [29] and Robotics [30], which can be deployed on a mobile device. Chair [31] is a generative model we use as a benchmark in this work.
3) *Autoencoders* are another class of neural networks used to learn a representation for a data set. Their applications are image reconstruction, image to image translation, and denoising to name a few. Mobile robots can be equipped with autoencoders to be used in their computer vision tasks. We use Pix2Pix [32], as a benchmark from this class.

3.2 Mobile and Server Setup

We used the Jetson TX2 module developed by NVIDIA® [33], a fair representation of mobile computation power as our mobile device. This module enables efficient implementation of DNN applications used in products such as robots, drones, and smart cameras. It is equipped with NVIDIA Pascal®GPU with 256 CUDA cores and a shared 8 GB 128 bit LPDDR4 memory between GPU and CPU. To measure the power consumption of the mobile platform, we used INA226 power sensor [34].

NVIDIA® Tesla® K40C [35] with 12 GB memory serves as our server GPU. The computation capability of this device is more than one order of magnitude compared to our mobile device.

TABLE 3
Mobile Networks Specifications in the US

Param.	3G	4G	Wi-Fi
Download speed (Mbps)	2.0275	13.76	54.97
Upload speed (Mbps)	1.1	5.85	18.88
α_u (mW/Mbps)	868.98	438.39	283.17
α_d (mW/Mbps)	122.12	51.97	137.01
β (mW)	817.88	1288.04	132.86

3.3 Communication Parameters

To model the communication between platforms, we used the average download and upload speed of mobile Internet [36], [37] for different networks (3G, 4G and Wi-Fi) as shown in Table 3.

The communication power for download (P_d) and upload (P_u) is dependent on the network throughput (t_d and t_u). Comprehensive examinations in [38] indicates that uplink and downlink power can be modeled with linear equations (Eq. (21)) fairly accurate with less than 6 percent error rate. Table 3 shows the parameter values of this equation for different networks.

$$\begin{aligned} P_u &= \alpha_u t_u + \beta \\ P_d &= \alpha_d t_d + \beta. \end{aligned} \tag{21}$$

4 Results

The latency and energy improvements of inference and online training with our engine for 8 different benchmarks are shown in Figs. 8 and 9, respectively. We considered the best case of mobile-only and cloud-only as our baseline. JointDNN can achieve up to 66 and 86 percent improvements in latency and energy consumption, respectively during inference. Communication cost increases linearly with batch size while this is not the case for computation cost and it grows with a much lower rate, as depicted in Fig. 10b. Therefore, a key observation is that as we increase the batch size, the mobile-only approach becomes more preferable.

During online training, the huge communication overhead of transmitting the updated weights will be added to the total cost. Therefore, to avoid downloading this large data, only a few back-propagation steps are computed in the cloud server. We performed a simulation by varying the percentage of updated weight. As the percentage of updated weights increases, the latency and energy consumption becomes constant which is shown in Fig. 10. This is the result of the fact that all the backpropagations will be performed on the mobile device and weights are not transferred from the cloud to the mobile. JointDNN can achieve

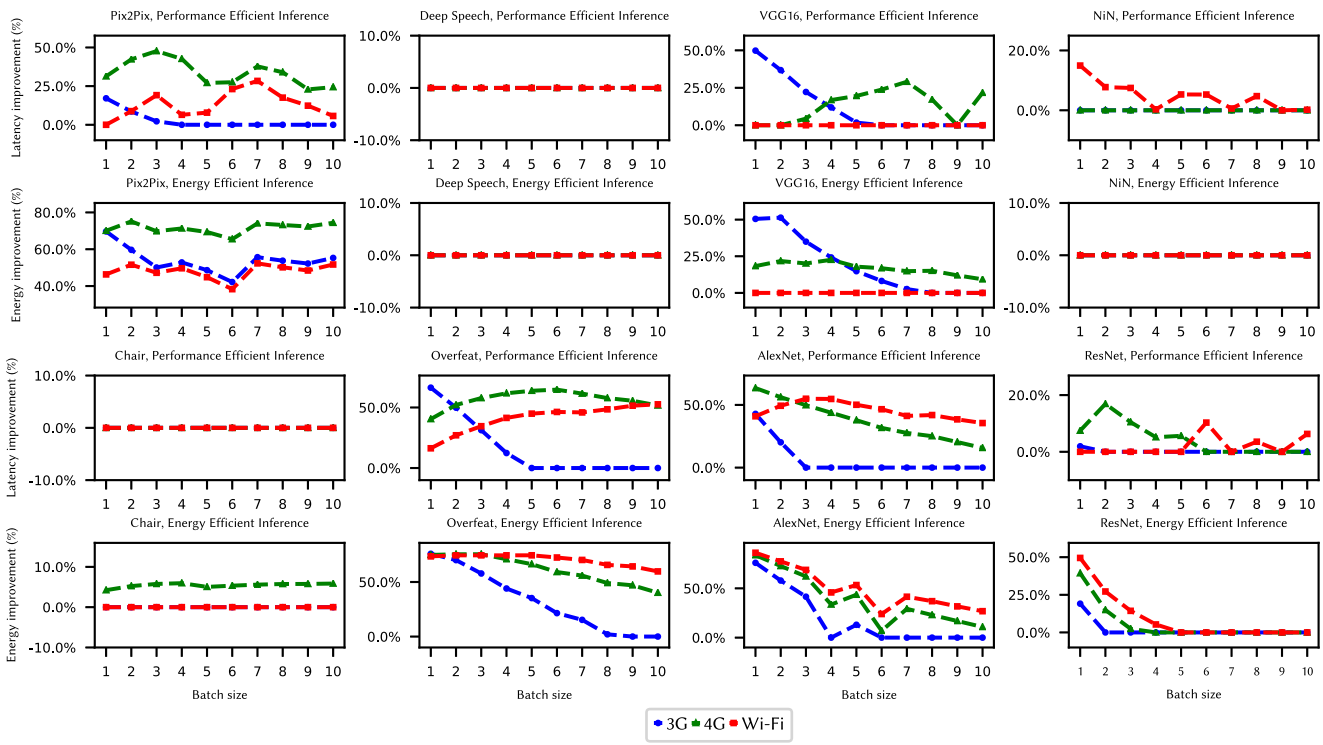

Fig. 8. Latency and energy improvements for different batch sizes during inference over the base case of mobile-only and cloud-only approaches.

improvements up to 73 percent in latency and 56 percent in energy consumption during inference.

Different patterns of scheduling are demonstrated in Fig. 11. They represent the optimal solution in the Wi-Fi network while optimizing for latency while mobile/cloud is allowed to use up to half of their computing resources. They show how the computations in DNN is divided between the mobile and the cloud. As can be seen, discriminative models (e.g., AlexNet), inference follows a mobile-cloud pattern and training follows a mobile-cloud-mobile pattern. The intuition is that the last layers are computationally intensive (fully connected layers) but with small data sizes, which require a low communication cost, therefore, the last layers tend to be computed on the cloud. For

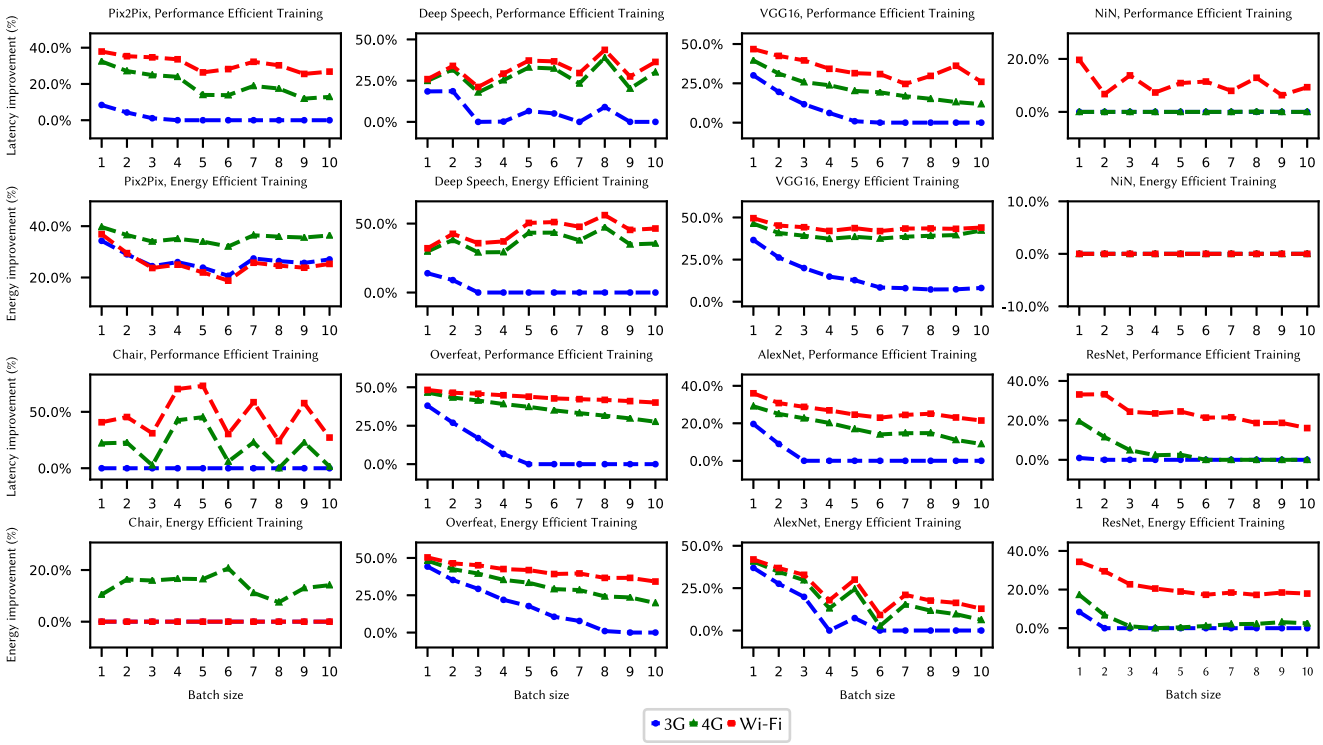

Fig. 9. Latency and energy improvements for different batch sizes during training over the base case of mobile-only and cloud-only approaches.

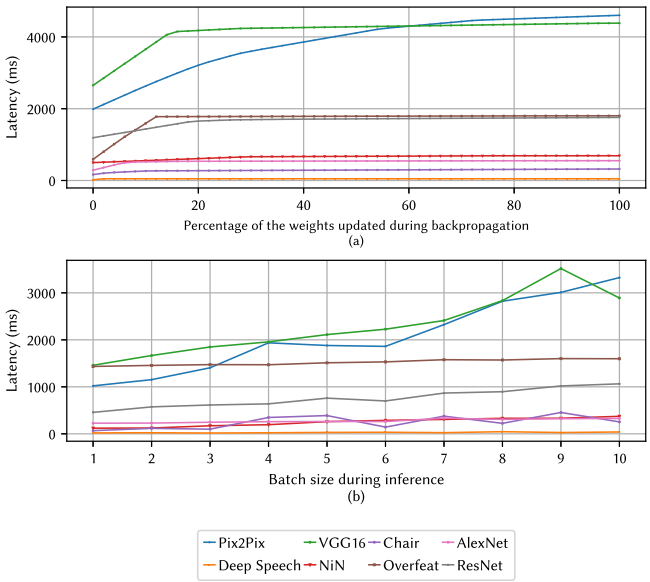

Fig. 10. (a) Latency of one epoch of online training using JointDNN algorithm versus percentage of updated weights. (b) Latency of mobile-only inference versus batch size.

generative models (e.g., Chair), the execution schedule of inference is the opposite of discriminative networks, in which the last layers are generally huge and in the optimal solution they are computed on the mobile. The reason behind not having any improvement over the base case of mobile-only is that the amount of transferred data is large. Besides, cloud-only becomes the best solution when the amount of transferred data is small (e.g., generative models). Lastly, for autoencoders, where both the input and output data sizes are large, the first and last layers are computed on the mobile.

JointDNN pushes some parts of the computations toward the mobile device. As a result, this will lead to less workload on the cloud server. As we see in Table 4, we can reduce the cloud server's workload up to 84 and 53 percent on average, which enables the cloud provider to provide service to more users, while obtaining higher performance and lower energy consumption compared to single-platform approaches.

4.1 Communication Dominance

Execution time and energy breakdown for AlexNet, which is noted as a representative for the state-of-the-art architectures deployed in cloud servers, is depicted in Fig. 12. The cloud-only approach is dominated by the communication costs. As demonstrated in Fig. 12, 99, 93 and 81 percent of the total execution time are used for communication in case of 3G, 4G, and Wi-Fi, respectively. This relative portion also

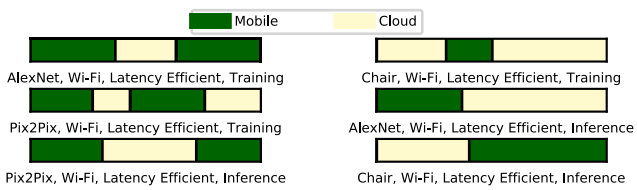

Fig. 11. Interesting schedules of execution for three types of DNN architectures while mobile/cloud are allowed to use up to half of their computing resources.

TABLE 4
Workload Reduction of the Cloud Server in Different Mobile Networks

Optimization Target	3G (%)	4G (%)	Wi-Fi (%)
Latency	84	49	12
Energy	73	49	51

applies to energy consumption. Comparing the latency and energy of the communication to those of mobile-only approach, we notice that the mobile-only approach for AlexNet is better than the cloud-only approach in all the mobile networks. We apply loss-less compression methods to reduce the overheads of communication, which will be covered in the next section.

4.2 Layer Compression

The preliminary results of our experiments show that more than 75 percent of the total energy and delay cost in DNNs are caused by communication in the collaborative approach. This cost is directly proportional to the size of the layer being downloaded to or uploaded from the mobile device. Because of the complex feature extraction process of DNNs, the size of some of the intermediate layers are even larger than the network's input data. For example, this ratio can go as high as $10\times$ in VGG16. To address this bottleneck, we investigated the compression of the feature data before any communication. This process can be applied to different DNN architecture types; however, we only considered CNNs due to their specific characteristics explained later in detail.

CNN architectures are mostly used for image and video recognition applications. Because of the spatially local preservation characteristics of *conv* layers, we can assume that the outputs of the first convolution layers are following the same structure as the input image, as shown in Fig. 13. Moreover, a big ratio of layer outputs is expected to be zero due to

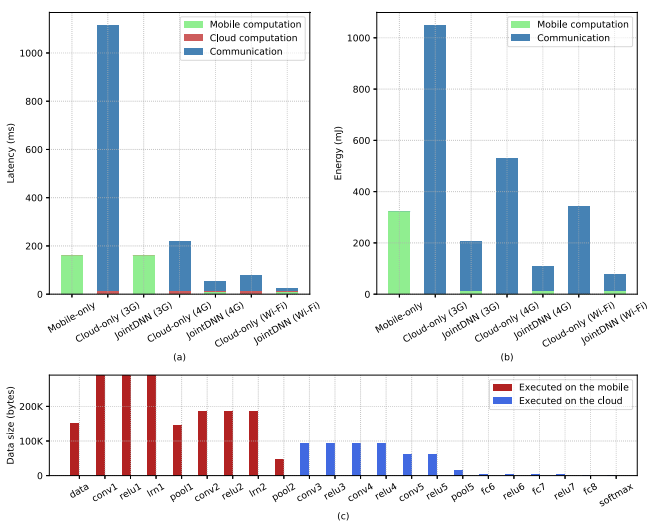

Fig. 12. (a) Execution time of AlexNet optimized for performance. (b) Mobile energy consumption of AlexNet optimized for energy. (c) Data size of the layers in AlexNet and the scheduled computation, where the first nine layers are computed on the mobile and the rest on the cloud, which is the optimal solution w.r.t. both performance and energy.

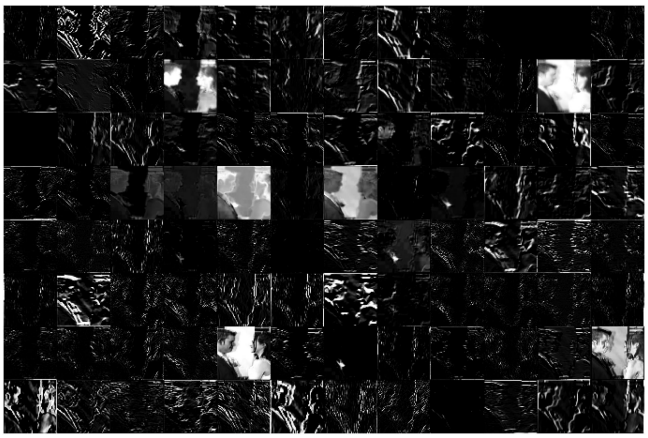

Fig. 13. Layer output after passing the input image through convolution, normalization and ReLU [39] layers. Channels are preserving the general structure of the input image and large ratio of the output data is black (zero) due to existence of *relu*. Tiling is used to put all 96 channels together.

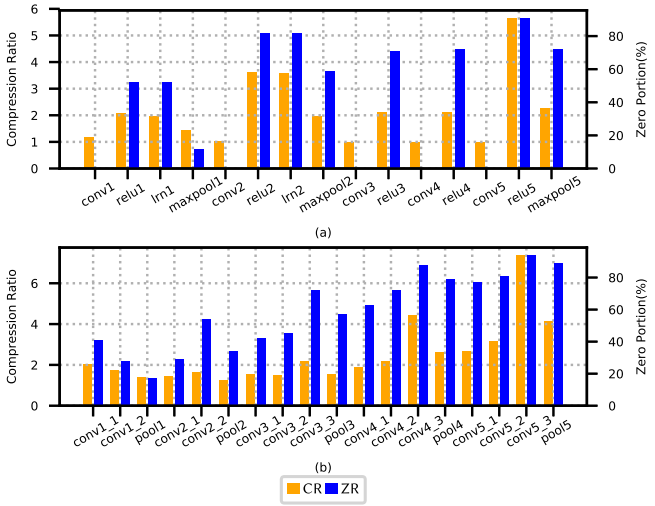

Fig. 14. Compression Ratio (CR) and ratio of zero valued neurons (ZR) for different layers of (a) AlexNet and (b) VGG16.

the presence of the ReLU layer. Our observations shows that the ratio of neurons equal to zero *(ZR)* varies from 50 to 90 percent after *relu* in CNNs. These two characteristics, layers being similar to the input image, and a large proportion of their data being a single value, suggest that we can employ existing image compression techniques to their output.

There are two general categories of compression techniques, lossy and loss-less [40]. In loss-less techniques, the exact original information is reconstructed. On the contrary, lossy techniques use approximations and the original data cannot be reconstructed. In our experiments, we examined the impact of compression of layer outputs using PNG, a loss-less technique, based on the encoding of frequent sequences in an image.

Even though the data type of DNN parameters in typical implementations is 32-bits floating-points, most image formats are based on 3-bytes RGB color triples. Therefore, to compress the layer in the same way as 2D pictures, the floating-point data should be quantized into 8-bits fixed-point. Recent studies show representing the parameters of DNNs with only 4-bits affects the accuracy, not more than 1 percent [5]. In this work, we implemented our architectures with an 8-bits fixed-point and presented our baseline without any compression and quantization. The layers of CNN contain numerous channels of 2D matrices, each similar to an image. A simple method is to compress each channel separately. In addition to extra overhead of file header for each channel, this method will not take the best of the frequent sequence decoding of PNG. One alternative is locating different channels side by side, referred to as tiling, to form a large 2D matrix representing one layer as shown in Fig. 13. It should be noted that 1D fully connected layers are very small and we did not apply compression on them.

The Compression Ratio *(CR)* is defined as the ratio of the size of the layer (8-bit) to the size of the compressed 2D matrix in PNG. Looking at the results of compression for two different CNN architectures in Fig. 14, we can observe a high correlation between the ratio of pixels being zero *(ZR)* and *CR*. PNG can compress the layer data up to $5.8\times$ and $3.5\times$ by average, therefore the communication costs can be reduced drastically. By replacing the compressed layer's output and adding the cost of the compression process itself, which is negligible compared to DNN operators, in JointDNN formulations, we achieve an extra $4.9\times$ and $4.6\times$ improvements in energy and latency on average, respectively.

5 Related Work and Comparison

General Task Offloading Frameworks. There are existing prior arts focusing on offloading computation from the mobile to the cloud[15], [41], [42], [43], [44], [45], [46]. However, all these frameworks share a limiting feature that makes them impractical for computation partitioning of the DNN applications.

These frameworks are programmer annotations dependent as they make decisions about pre-specified functions, whereas JointDNN makes scheduling decisions based on the model topology and mobile network specifications in run-time. Offloading in function level, cannot lead to efficient partition decisions due to layers of a given type within one architecture can have significantly different computation and data characteristics. For instance, a specific convolution layer structure can be computed on mobile or cloud in different models in the optimal solution.

Neurosurgeon [14] is the only prior art exploring a similar computation offloading idea in DNNs between the mobile device and the cloud server at layer granularity. Neurosurgeon assumes that there is only one data transfer point and the execution schedule of the efficient solution starts with mobile and then switches to the cloud, which performs the whole rest of the computations. Our results show this is not true especially for online training, where the optimal schedule of execution often follows the mobile-cloud-mobile pattern. Moreover, generative and autoencoder models follow a multi-transfer points pattern. Also, the execution schedule can start with the cloud especially in case of generative models where the input data size is large. Furthermore, inter-layer optimizations performed by DNN libraries are not considered in Neurosurgeon. Moreover, Neurosurgeon only schedules for optimal latency and energy, while JointDNN adapts to different scenarios including battery limitation, cloud server congestion, and QoS. Lastly,

Neurosurgeon only targets simple CNN and ANN models, while JointDNN utilizes a graph-based approach to handle more complex DNN architectures like ResNet and RNNs.

6 Conclusions and Future Work

In this paper, we demonstrated that the status-quo approaches, cloud-only or mobile-only, are not optimal with regard to latency and energy. We reduced the problem of partitioning the computations in a DNN to shortest path problem in a graph. Adding constraints to the shortest path problem makes it NP-Complete, therefore, we also provided ILP formulations to cover different possible scenarios of limitations of mobile battery, cloud congestion, and QoS. The output data size in discriminative models is typically smaller than other layers in the network, therefore, last layers are expected to be computed on the cloud, while first layers are expected to be computed on the mobile. Reverse reasoning works for Generative models. Autoencoders have large input and output data sizes, which implies that the first and last layers are expected to be computed on the mobile. With these insights, the execution schedule of DNNs can possibly have various patterns depending on the model architecture in model cloud computing. JointDNN formulations are designed for feed-forward networks and its extension to recurrent neural networks will be studied as a future work.

Acknowledgments

This research was supported by grants from NSF SHF, DARPA MTO, and USC Annenberg Fellowship.

References

[1] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 92:1–92:36, Sep. 2018. [Online]. Available: http://doi.acm.org/10.1145/3234150

[2] K.-S. Oh and K. Jung, "GPU implementation of neural networks," *Pattern Recognit.*, vol. 37, pp. 1311–1314, Jun. 2004.

[3] 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," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999271

[4] M. S. Razlighi, M. Imani, F. Koushanfar, and T. Rosing, "LookNN: Neural network with no multiplication," in *Proc. Design Autom. Test Europe Conf. Exhib.*, March 2017, pp. 1775–1780.

[5] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[6] K. Skala, D. Davidovic, E. Afgan, I. Sovic, and Z. Sojat, "Scalable distributed computing hierarchy: Cloud, fog and dew computing," *Open J. Cloud Comput.*, vol. 2, no. 1, pp. 16–24, 2015. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:101:1-201705194519

[7] A. Newsroom, "The future is here: iPhone X," 2017. [Online]. Available: https://www.apple.com/newsroom/2017/09/the-future-is-here-iphone-x/, Accessed: Jan. 15, 2018.

[8] H. Li, J. Yu, Y. Ye, and C. Bregler, "Realtime facial animation with on-the-fly correctives," *ACM Trans. Graphics*, vol. 32, no. 4, pp. 42:1–42:10, Jul. 2013.

[9] A. E. Eshratifar, A. Esmaili, and M. Pedram, "BottleNet: A deep learning architecture for intelligent mobile cloud computing services," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design*, Jul. 2019, pp. 1–6.

[10] A. E. Eshratifar, A. Esmaili, and M. Pedram, "Towards collaborative intelligence friendly architectures for deep learning," in *Proc. 20th Int. Symp. Qual. Electron. Design*, Mar. 2019, pp. 14–19.

[11] A. E. Eshratifar and M. Pedram, "Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment," in *Proc. Great Lakes Symp. VLSI*, 2018, pp. 111–116. [Online]. Available: http://doi.acm.org/10.1145/3194554.3194565

[12] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," in *Robot. Sci. Syst.*, 2018.

[13] M. Nazemi, A. E. Eshratifar, and M. Pedram, "A hardware-friendly algorithm for scalable training and deployment of dimensionality reduction models on FPGA," in *Proc. 19th IEEE Int. Symp. Qual. Electron. Design*, 2018, pp. 395–400.

[14] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2017, pp. 615–629.

[15] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. 6th Conf. Comput. Syst.*, 2011, pp. 301–314.

[16] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 328–339.

[17] R. W. Ahmad, A. Gani, S. H. A. Hamid, F. Xia, and M. Shiraz, "A review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues," *J. Netw. Comput. Appl.*, vol. 58, pp. 42–59, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804515002088

[18] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014. [Online]. Available: http://arxiv.org/abs/1410.0759

[19] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[20] S. Hong and H. Kim, "An integrated GPU power and performance model," *SIGARCH Comput. Architecture News*, vol. 38, no. 3, pp. 280–289, Jun. 2010.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. 2016 IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 770–778.

[22] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE J. Sel. Areas Commun.*, vol. 14, no. 7, pp. 1228–1234, Sep. 1996.

[23] A. Juttner, B. Szviatovski, I. Mecs, and Z. Rajko, "Lagrange relaxation based method for the QoS routing problem," in *Proc. IEEE Conf. Comput. Commun.*, 2001, vol. 2, pp. 859–868.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[25] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated recognition, localization and detection using convolutional networks," *CoRR*, 2013. [Online]. Available: http://arxiv.org/abs/1312.6229

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[27] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep speech: Scaling up end-to-end speech recognition," *CoRR*, vol. abs/1412.5567, 2014. [Online]. Available: http://arxiv.org/abs/1412.5567

[28] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, 2013. [Online]. Available: http://arxiv.org/abs/1312.4400

[29] I. Goodfellow, J. Pouget-Abadie , M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[30] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *Proc. 2017 IEEE Int. Conf. Robot. Automat.*, 2016, pp. 2786–2793.

[31] A. Dosovitskiy, J. T. Springenberg, and T. Brox, "Learning to generate chairs with convolutional neural networks," *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1411.5928

[32] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1611.07004
[33] N. Corporation, "Jetson TX2 Module," 2018. [Online]. Available: https://developer.nvidia.com/embedded/buy/jetson-tx2, Accessed: Jan. 15, 2018.
[34] T. I. Incorporated, "INA current/power monitor," 2018. [Online]. Available: http://www.ti.com/product/INA226, Accessed: Jan. 15, 2018.
[35] N. Corporation, "Tesla data center GPUs for servers," 2018. [Online]. Available: http://www.nvidia.com/object/tesla-servers.html, Accessed: Jan. 15, 2018.
[36] OpenSignal.com, "State of mobile networks: USA," 2017. [Online]. Available: https://opensignal.com/reports/2017/08/usa/state-of-the-mobile-network, Accessed: Jan. 15, 2018.
[37] OpenSignal.com, "United States speedtest market report," 2017. [Online]. Available: http://www.speedtest.net/reports/united-states/ Accessed: Jan. 15, 2018.
[38] 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 *Proc. 10th Int. Conf. Mobile Syst. Appl. Services*, 2012, pp. 225–238.
[39] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, Apr. 2011, pp. 315–323. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a.html
[40] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Hoboken, NJ, USA: Wiley-Interscience, 2006.
[41] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in *Proc. 9th Int. Conf. Mobile Syst. Appl. Services*, 2011, pp. 43–56. [Online]. Available: http://doi.acm.org/10.1145/1999995.2000000
[42] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "COMET: Code offload by migrating execution transparently," in *Proc. 10th USENIX Conf. Operating Syst. Design Implementation*, 2012, pp. 93–106. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387890
[43] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Services*, 2010, pp. 49–62. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814441
[44] X. Wang, X. Liu, Y. Zhang, and G. Huang, "Migration and execution of JavaScript applications between mobile devices and cloud," in *Proc. 3rd Annu. Conf. Syst. Program. Appl.: Softw. Humanity*, 2012, pp. 83–84.
[45] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang, "Refactoring Android Java code for on-demand computation offloading," *ACM SIGPLAN Notices*, vol. 47, no. 10, pp. 233–248, Oct. 2012.
[46] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Netw. Appl.*, vol. 18, no. 1, pp. 129–140, Feb. 2013. [Online]. Available: https://doi.org/10.1007/s11036-012-0368-0

Amir Erfan Eshratifar received the dual BS degrees in electrical engineering and computer science from the Sharif University of Technology, Tehran, Iran, in 2017. He is currently working toward the PhD degree in the Ming Hsieh Department of Electrical Engineering, University of Southern California (USC), Los Angeles, CA.

Mohammad Saeed Abrishami received the BS degree in electrical engineering from the University of Tehran, Tehran, Iran, in 2014. He is currently working toward the PhD degree in the Ming Hsieh Department of Electrical Engineering, University of Southern California (USC), Los Angeles, CA.

Massoud Pedram (F'01) received the BS degree in electrical engineering from the California Institute of Technology, Pasadena, CA, in 1986, and the MS and PhD degrees in electrical engineering and computer sciences from the University of California Berkeley, CA, in 1989 and 1991, respectively. In 1991, he joined the Ming Hsieh Department of Electrical Engineering, University of Southern California (USC), Los Angeles, CA, where he is currently the Charles Lee Powel professor of the USC Viterbi School of Engineering. He is a fellow of the IEEE.