Energy and Performance Efficient Computation Offloading for Deep Neural Networks in a Mobile Cloud Computing Environment

Amir Erfan Eshratifar
Dept. of Electrical Engineering
University of Southern California
Los Angeles, California
eshratif@usc.edu

Massoud Pedram
Dept. of Electrical Engineering
University of Southern California
Los Angeles, California
pedram@usc.edu

ABSTRACT

In today's computing technology scene, mobile devices are considered to be computationally weak, while large cloud servers are capable of handling expensive workloads, therefore, intensive computing tasks are typically offloaded to the cloud. Recent advances in learning techniques have enabled Deep Neural Networks (DNNs) to be deployed in a wide range of applications. Commercial speech based intelligent personal assistants (IPA) like Apple's Siri, which employs DNN as its recognition model, operate solely over the cloud. The cloud-only approach may require a large amount of data transfer between the cloud and the mobile device. The mobile-only approach may lack performance efficiency. In addition, the cloud server may be slow at times due to the congestion and limited subscription and mobile devices may have battery usage constraints. In this paper, we investigate the efficiency of offloading only some parts of the computations in DNNs to the cloud. We have formulated an optimal computation offloading framework for forward propagation in DNNs, which adapts to battery usage constraints on the mobile side and limited available resources on the cloud. Our simulation results show that our framework can achieve 1.42× on average and up to 3.07× speedup in the execution time on the mobile device. In addition, it results in 2.11× on average and up to 4.26× reduction in mobile energy consumption.

KEYWORDS

computation offloading, mobile cloud computing, deep neural networks, energy efficient computing, high performance computing

ACM Reference Format:
Amir Erfan Eshratifar and Massoud Pedram. 2018. Energy and Performance Efficient Computation Offloading for Deep Neural Networks in a Mobile Cloud Computing Environment. In *GLSVLSI '18: 2018 Great Lakes Symposium on VLSI, May 23–25, 2018, Chicago, IL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3194554.3194565

GLSVLSI '18, May 23–25, 2018, Chicago, IL, USA

ACM ISBN 978-1-4503-5724-1/18/05…$15.00
https://doi.org/10.1145/3194554.3194565

1 INTRODUCTION

With the exponential growth of deep learning techniques and digital raw data, a wide range of applications, that were previously intractable, are enjoying scalable and easy-to-train models. Deep learning has shown extremely promising results in Computer Vision[1][2], Speech Recognition[3], and Natural Language Processing [4]. Even though, DNNs outperform many other machine learning models, they often require a great amount of computational resources. Mobile devices like smart-phones and robots are often considered limited in terms of processing resources and energy budget compared to the cloud server, therefore, DNN workloads are often entirely offloaded to the cloud. However, as the computational resources in the mobile devices are becoming more powerful and energy efficient, we could possibly push some parts of the computations toward the mobile edge. Basically, there are following methods during neural networks inferences in mobile cloud computing context:

Cloud-only approach: In this method, when the user makes a query, all the input data will be uploaded to the cloud server, afterwards the cloud server does the inference and sends the output back to the mobile. The main drawback of this method is the communication overheads of uploading large inputs (e.g., images, audios, videos), which includes latency and energy costs or downloading large outputs in case of generative models, where the output of the network could be a multimedia file (e.g., image in [7]). The latency and energy costs of several benchmarks over three different wireless networks (Wi-Fi, 3G, LTE) are studied comprehensively in section (3). This approach also compromises user's privacy.

Mobile-only approach: In this method, all the computations are performed locally in either CPU or GPU depending on the available hardware and software architectures. There is an active research in empowering mobile devices for DNNs by designing accelerators[34][33]. Recently, Apple has unveiled a new machine learning framework API for developers named Core ML, enabling on-device processing[8]. They have also included a customized GPU specialized for machine learning tasks in their new smartphone[26]. In addition, by using this approach, no data will leave the device, which provides more privacy for the user.

Mobile-cloud collaborative approach: In this method, we seek to develop an optimal scheduling algorithm for collaboratively computation of feed forward neural networks. We break down the computations at layer granularity and decide upon which layers should be computed on mobile and which on the cloud server to achieve maximum performance and energy efficiency. Prior arts in

this area often require to transfer the whole process state, which could have potentially large amount of variables to be offloaded, that is not taken into account[29]. Some rule-based offloading frameworks offload a function, when it is exceeding a pre-defined time ignoring the amount of data to be offloaded[31][30]. In addition, statistical prediction of the execution time and energy of functions used in [28] are prone to large errors especially in case of DNNs (Section 2). Our main contribution in this work is formulating optimal computation offloading in DNNs at layer granularity, where the optimality comes from solving an integer linear programming (ILP) setup.

There are three approaches in estimating the performance and energy consumption of inference with neural networks:

1. Statistical modeling by varying the configurable parameters of each layer and measuring the latency and energy consumption for each set of input parameters. A regression model can be used to fit over these profiles so as to estimate the latency and energy of the layer based on its configuration[27]. In section 2, we show that the complexities in latency and energy estimation of the conventional layers in deep neural networks are quite high so that it is not feasible to use statistical modeling, because we need exponential number of profiling measurements.

2. Analytical modeling by considering hardware and software specifications. State-of-the-art work in latency modeling of deep neural networks [5] fails to estimate fine-grained layer-level latency within an acceptable error bound. For instance, the latency of 6th layer in VGG-16, which is a fully connected layer with 4096 neurons, is underestimated by around 900 percent. Because of the fact that manufacturers do not reveal the detailed hardware architecture specifications in addition to the proprietary parallel computing architectures like CUDA, analytical approach could be quite challenging[6].

3. Application-specific profiling approach only requires to profile the latency and energy of the target neural networks. Because of the limited number of applications, which requires neural networks in user's mobile, this method seems more practical and leads to more accurate results.

In summary, the main contribution of this work is providing a framework to query DNNs efficiently on mobile devices collaborating with cloud server in the presence of battery usage constraints and cloud server limited resources.

2 EXECUTION TIME AND ENERGY PROFILING

In this section we elaborate upon the challenges in execution time and energy profiling. We performed four sets of experiments with the convolution operator, which is generally the most expensive operator in neural networks, on our mobile platform using Tensorflow™1.0.1[9]. The convolution takes a batch of images of width *img_w*, height *img_h*, and with *img_c* input channels. The image is convolved with a filter of width f_w, height f_h, and f_c output channels with a pre-defined stride in x and y directions. The input sizes in our experiments are listed in table 1. As depicted in Figure 1, at some points, increasing the input size decreases the execution time, which is caused by the heuristic selection of different convolution algorithms available in cuDNN[10]. For instance, for

Table 1: Convolution inputs size

	batches	img_w	img_h	img_c	f_w	f_h	f_c
Case1	16	64	64	Var	3	3	256
Case2	16	64	64	Var	9	9	256
Case3	16	64	64	3	Var	Var	256
Case4	16	64	64	128	Var	Var	256

small filter sizes, the convolution is calculated using General Matrix Multiplication (GEMM), while for large filter sizes an algorithm which uses Fast Fourier Transform (FFT) is chosen. Another source of non-linearity is when the GPU is under-utilized and increasing the input size would not affect the execution time. Moreover, as

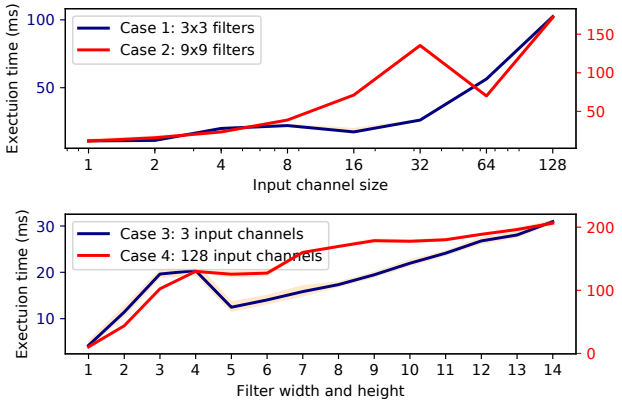

Figure 1: Non-linear execution time of convolutions w.r.t. inputs size. The execution time decreases even when the input size increases because of the heuristic selection of convolution algorithms.

depicted in Figure 2, layer-wise profiling is prone to large amount of errors in estimation. The reason is the accelerations done by hardware and software frameworks for the consecutive operations. For instance, the latency of two consecutive convolution $A * A * x$ is generally less than the sum of each convolution. The main reason is that the order of grouping of the operators affects the latency. In other words, the latency of $(A * A) * x$ and $A * (A * x)$ are not the same. In a nutshell, the execution time of two consecutive layers may not even be close to the sum of the execution times of the individual layers. This fact is depicted in Figure 2 for AlexNet[1]. Therefore, for a statistical model we need to profile not only each layer type with different input sizes but also the combination of various layers, which makes the act of profiling and regression very time-consuming and impractical. As a result, we decide to perform application-specific profiling. The level of granularity we are considering in our profiling step, is the following list of the layer types: convolution, deconvolution, pooling, fully connected, and activation functions.

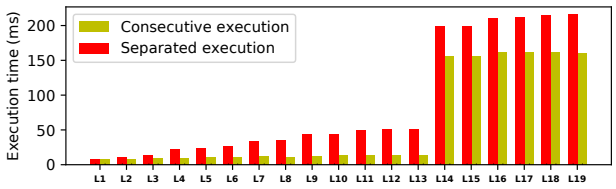

Figure 2: Non-linearity in the execution time of consecutive layers in AlexNet[1]. Each bar represents the execution time up to that layer.

3 COMPUTATION-OFFLOADING MODEL FOR DNN

3.1 Performance Efficient Computation Offloading Setup

As a parallel work [32] shows, mobile cloud computing scheduling of different tasks with constraints is a NP-hard problem. Therefore, we formulated this problem as an ILP with tractable number of variables. In our method, first we profile the latency and energy consumption of consecutive layers of size $m \in \{1, 2, \ldots, num\_layers\}$. We define the number of layers to be n consistently throughout this paper. Thus, we will have

$$n + (n-1) + \ldots + 1 = n(n+1)/2 \tag{1}$$

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

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

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

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

$T_{mobile_{L_{i,j}}}$ and $T_{cloud_{L_{i,j}}}$ represent the execution time of ith layer to 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 ith layer respectively. Considering each set of consecutive layers, whenever $m_{i,j}$ and one of $\{c_{j+1,k}\}_{k=j+1:n}$ are one, output of jth is uploaded to the cloud. The same argument works for download. We also note that the last two terms in (3) represent the condition, when last layer is computed on the cloud and we need to download the output to the mobile edge and first layer is computed on the cloud and we need to upload the input to the cloud respectively. Because each set of consecutive layers should be computed either on mobile or cloud or none, we define the following set of constraints:

$$\forall i \in 1:n, \forall j \in i:n : m_{i,j} + c_{i,j} \leq 1 \tag{5}$$

Besides, we have to make sure that all layers are being computed exactly once. Therefore, we need to add the following constraints:

$$\forall m \in 1:n : \sum_{i=1}^{m}\sum_{j=m}^{n}(m_{i,j} + c_{i,j}) = 1 \tag{6}$$

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

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

with the following constraints:

$$\begin{aligned} &u_{i,j} \leq m_{i,j} \\ &u_{i,j} \leq \sum_{k=j+1}^{n} c_{j+1,k} \\ &m_{i,j} + \sum_{k=j+1}^{n} c_{j+1,k} - u_{i,j} \leq 1 \\ &d_{i,j} \leq c_{i,j} \\ &d_{i,j} \leq \sum_{k=j+1}^{n} m_{j+1,k} \\ &c_{i,j} + \sum_{k=j+1}^{n} m_{j+1,k} - d_{i,j} \leq 1 \end{aligned} \tag{8}$$

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.

3.2 Energy Efficient Computation Offloading Setup

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

$$E_{computation} = \sum_{i=1}^{n}\sum_{j=i}^{n} m_{i,j}.E_{mobile_{L_{i,j}}} \tag{9}$$

$$E_{communication} = \sum_{i=2}^{n}\sum_{j=i}^{n} m_{i,j}.E_{download_{L_i}} + \sum_{i=1}^{n}\sum_{j=i}^{n-1} m_{i,j}.E_{upload_{L_j}} + (\sum_{i=1}^{n}(1-m_{1,i}) - (n-1)).E_{upload_{L_1}} + (\sum_{i=1}^{n}(1-m_{i,n}) - (n-1)).E_{download_{L_n}} \quad (10)$$

$$E_{total} = E_{computation} + E_{communication} \quad (11)$$

$E_{mobile_{L_{i,j}}}$ and $E_{cloud_{L_{i,j}}}$ represent the amount of energy required to compute ith layer to jth layer on the mobile and cloud respectively. $E_{download_{L_i}}$ and $E_{upload_{L_i}}$ represent the amount of energy required to download and upload the output of ith layer respectively. Likewise for performance efficient ILP constraints, we need to make sure that each layer get executed exactly once:

$$\forall m \in 1:n : \sum_{i=1}^{m}\sum_{j=m}^{n} m_{i,j} \leq 1 \quad (12)$$

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.

3.3 Scenarios

- **Performance Efficient Computation:** In this case, all we need to do is to solve the ILP formulation for performance efficient computation offloading.
- **Energy Efficient Computation:** In this case, all we need to do is to solve the ILP formulation for energy efficient computation offloading.
- **Battery Budget Limitation:** In this case, according to the available battery, the operating system can decide to dedicate a specific amount of energy usage to each application. By adding the following constraint to the performance efficient ILP formulation, our framework would adapt to battery limitations:

$$E_{computation} + E_{communication} \leq E_{ubound} \quad (13)$$

- **Cloud Limited Resources:** There are situations in which the cloud server is busy or the subscription of the user is limited so that a limit of execution time could be applied to each application to alleviate the server load:

$$\sum_{i=1}^{n}\sum_{j=i}^{n} c_{i,j}.T_{cloud_{L_{i,j}}} \leq T_{ubound} \quad (14)$$

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

4 CASE STUDIES

4.1 Experimental Setup

We use Jetson TX2 board developed by NVIDIA[16], which fairly represents the computing power of mobile devices. Our server platform is equipped with NVIDIA®Tesla ™K40C[15], which has 11.25x more computing resources than our mobile platform. We measure the GPU power on our mobile platform using INA226 power monitoring sensor with sampling rate of 500kHz[19]. In our experiment, we use Tensorflow™[9], which is an open source software library equipped with cuDNN[10], a GPU-accelerated library of primitives for deep neural networks. Average download and upload speed of mobile networks in the U.S. are considered in this experiment[11][12]. We also use the power models of [14] for mobile networks with error less than 6%. The power level for uplink is $P_u = \alpha_u t_u + \beta$ and for downlink is $P_d = \alpha_d t_d + \beta$, where t_u and t_d are uplink and downlink throughputs respectively. The values for our power model paramters are as the following table:

Table 2: Power model parameters

Network	α_u (mW/Mpbs)	α_d (mW/Mpbs)	β (mW)
3G	868.98	122.12	817.88
4G	438.39	51.97	1288.04
Wi-Fi	283.17	137.01	132.86

Table 3: Average download and upload speed of mobile networks in U.S.

Network	Download speed (Mpbs)	Upload speed (Mbps)
3G	2.0275	1.1
4G	13.76	5.85
Wi-Fi	54.97	18.88

4.2 Benchmarks

Since the architecture of neural networks depends on the type of the application, we have chosen three common applications of DNNs:

(1) Discriminative neural networks are a class of models in machine learning for modeling the conditional probability distribution $P(y|x)$. This essentially is used in classification and regression tasks. AlexNet[1], OverFeat[21], VGG16[22] and NiN[20] are the discriminative models we use as benchmarks in this experiment.
(2) Generative neural networks model the joint probability distribtion $P(x, y)$, which allows for generating new samples. They have applications in computer vision[17] and robotics [13], which can be deployed on a mobile device. Chair[23] is a generative model we use as benchmark.
(3) Autoencoders are a class of neural networks, which are used to learn a representation for a data set. Their application is in image reconstruction, image to image translation, and denoising to name a few. Mobile robots can be equipped with autoencoders for their computer vision tasks.

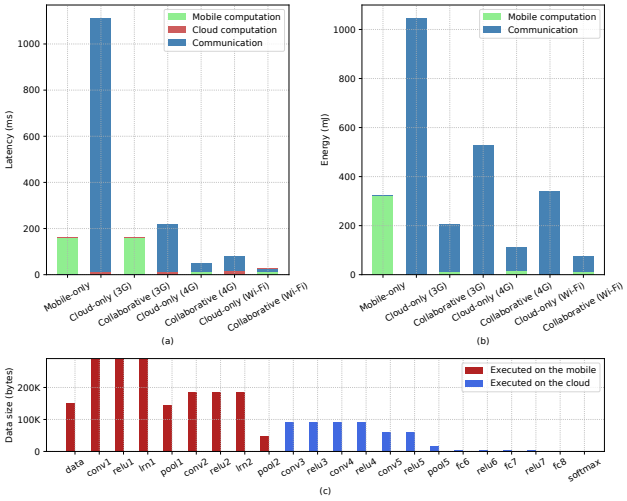

Figure 3: Execution time of AlexNet optimized for performance (a) Mobile energy consumption of AlexNet optimized for energy (b) Data size of the layers in AlexNet and the scheduled computation, where the first seven layers are being computed on the mobile and the rest on the cloud, which is optimal w.r.t. both performance and energy (c).

We use Pix2Pix[23], which has a similar architecture to autoencoders, as our benchmark.

Table 4: Benchmark Specifications

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

4.3 Results

Execution time and energy breakdown for AlexNet, which is noted as a representative for the state-of-the-art models deployed in cloud servers[25], is depicted in Figure 3. Not only is the cloud-only approach dominated by the communication costs but also the collaborative one. As demonstrated in Figure 3, 99%, 93% and 81% of the total execution time is used for communication in case of 3G, 4G, and Wi-Fi. This relative portion also applies to energy consumption. Comparing the latency and energy of the communication to those of mobile-only approach, we figure out that mobile-only approach for AlexNet is better than the cloud-only approach in all the mobile networks.

Figure 4 shows the execution time and energy improvements for 6 different benchmarks. We have considered the best case of cloud-only and mobile-only as our baseline. We have also depicted the schedule of computations in this figure. As we can see, the cloud-only and mobile-only approaches are optimal for VGG16 and NiN respectively, while optimizing for mobile energy consumption. Our proposed framework can achieve 1.42X on average and up to 3.07X (OverFeat, 3G) speedup when optimizing for performance. Mobile energy consumption can be reduced 2.11X on average and up to 4.26X (OverFeat, 3G), while optimizing for energy consumption. The improvements in the execution time of deep neural networks benefits not only the mobile users but also the cloud providers, which essentially means that they could save more resources or have more users. Another key observation is the scheduling pattern of the computations in different types of neural networks. Generally, in discriminative networks, the input size is often large and the output size is small, which implies that it is better to compute the first layers on the mobile to avoid uploading large inputs to the cloud. On the other hand, in generative networks, output size is often large, which implies that it is better to compute the last layers on the mobile to avoid downloading large outputs from the cloud. In autoencoders, both the input and output data sizes are often large, therefore, it is better to compute the first and last layers on the mobile and offload the computations in the middle layers to the cloud.

4.4 Cloud Limited Available Resources Experiment

The cloud resource management may decide to allow a process to be executed for a limited amount of time. The reason could be the user's limited subscription or congestion in the cloud server. In this scenario, we have set up a experiment in which the cloud allows up to 50% of the execution time of the whole neural network on the cloud over Wi-Fi network setup. As we have shown in the Figure 5, multiple data transfer points may occur, whose maximum is four in case of OverFeat.

CONCLUSION AND ROADMAP

In this work, we demonstrated that cloud-only or mobile-only approaches are not necessarily optimal with regard to performance and energy. We formulated the problem with ILP setup, which is solved only once by the cloud server. The reason that we are using ILP is the fact that the number of data transfer points strongly depends on the network architecture and resource constraints applied by the cloud and mobile. It is possible to limit the number of data transfer points and solve the problem by a polynomial algorithm, but it is not guaranteed to be optimal for all kinds of network architecture and constraints. The output data size of generative models, are generally large so the last layers are expected to be computed on the mobile in the optimal case to avoid downloading large data. On the other hand, classification networks have smaller output data size, therefore last layers are expected to be computed on the cloud. 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 computation scheduling of deep neural networks can possibly have different patterns depending on the model architecture. The proposed framework in this paper can be extended to any data flow computational graphs with the granularity of primitive operators like multiplication.

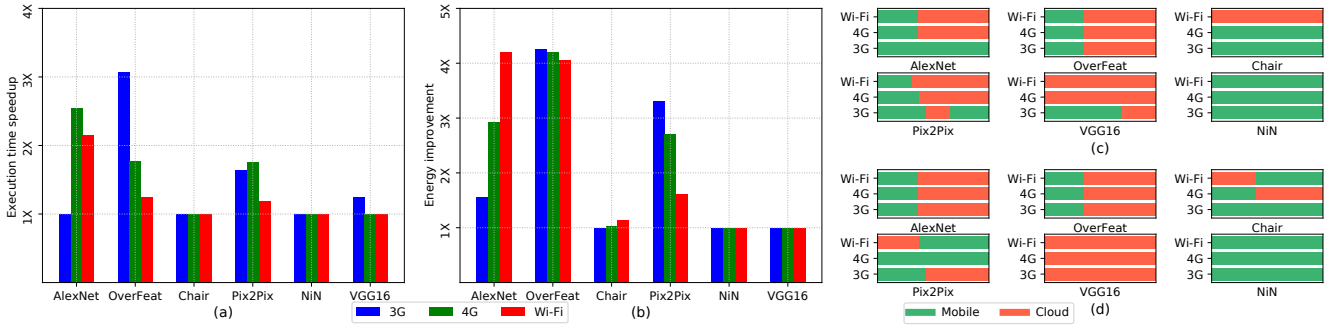

Figure 4: Execution time speed up (a) and energy improvement (b) using our proposed framework over the best case of mobile-only and cloud-only approaches. Scheduling patterns in performance efficient (c) and energy efficient (d) setups.

Figure 5: Computation scheduling patterns under execution time constraint of the cloud over Wi-Fi network setup

REFERENCES

[1] Krizhevsky et al., Imagenet classification with deep convolutional neural networks, NIPS, 2012.
[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. preprint arXiv:1409.1556, 2014.
[3] Hinton, G. et al. Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Processing Magazine 29, 2012.
[4] Collobert, R., et al. Natural language processing (almost) from scratch. J. Mach. Learn. Res. 12, 2493âĂŞ2537, 2011.
[5] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. Paleo: A Performance Model for Deep Neural Networks. International Conference on Learning Representations, 2017.
[6] Sunpyo Hong and Hyesoon Kim. An integrated GPU power and performance model. In Proceedings of the 37th annual international symposium on Computer architecture, 2010.
[7] Alec Radford, Luke Metz, Soumith Chintala: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. CoRR abs/1511.06434, 2015.
[8] Apple announces new machine learning API to make mobile AI faster https://www.theverge.com/2017/6/5/15725994/apple-mobile-ai-chip-announced-wwdc-2017
[9] M. Abadi, A. Agarwal et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems, arXiv:1603.04467, 2016.
[10] NVIDIA cuDNN https://developer.nvidia.com/cudnn
[11] State of Mobile Networks: USA - OpenSignal https://opensignal.com/reports/2017/08/usa/state-of-the-mobile-network
[12] 2017 United States Speedtest Market Report http://www.speedtest.net/reports/united-states/
[13] Finn, C. and Levine, S. Deep visual foresight for planning robot motion. arXiv preprint arXiv:1610.00696, 2016.
[14] Junxian Huang, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In Proceedings of the 10th international conference on Mobile systems, applications, and services, 2012.
[15] Tesla GPU Accelerators for Servershttp://www.nvidia.com/object/tesla-servers.html
[16] Jetson TX2 Module https://developer.nvidia.com/embedded/buy/jetson-tx2
[17] Ian J. Goodfellow, Jean Pouget-Abadie et al., âĂIJGenerative Adversarial Networks,âĂİ arXiv:1406.2661, 2014.
[18] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on International Conference on Machine Learning, 2010.
[19] INA226 http://www.ti.com/product/INA226
[20] Min Lin, Qiang Chen, Shuicheng Yan, Network In Network. arXiv preprint arXiv:1312.4400, 2014.
[21] Pierre Sermanet et al., OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. arXiv:1312.6229, 2014.
[22] Karen Simonyan, Andrew Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556, 2014.
[23] Alexey Dosovitskiy et al., Learning to Generate Chairs, Tables and Cars with Convolutional Networks. arXiv:1411.5928, 2017.
[24] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros, Image-to-Image Translation with Conditional Adversarial Networks. arXiv:1611.07004, 2016.
[25] Adam Coates, Brody Huval, Tao Wang, David J. Wu, Andrew Y. Ng, and Bryan Catanzaro. Deep learning with COTS HPC systems. In Proceedings of the 30th International Conference on International Conference on Machine Learning, 2013.
[26] The new iPhone 8 has a custom GPU designed by Apple with its new A11 Bionic chip https://techcrunch.com/2017/09/12/the-new-iphone-8-has-a-custom-gpu-designed-by-apple-with-its-new-a11-bionic-chip/
[27] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. SIGARCH Comput. Archit, 2017.
[28] Eduardo Cuervo et al., Maui: making smartphones last longer with code offload. In Proceedings of the 8th international conference on Mobile systems, applications, and services, ACM, 2010.
[29] Roelof Kemp Nicholas Palmer Thilo KielmannHenri Bal, Cuckoo: A Computation Offloading Framework for Smartphones, International Conference on Mobile Computing, Applications, and Services, 2010.
[30] Mark S Gordon, Davoud Anoushe Jamshidi, Scott A Mahlke, Zhuoqing Morley Mao, and Xu Chen. Comet: Code offload by migrating execution transparently, 2012.
[31] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. CloneCloud: elastic execution between mobile device and cloud. In Proceedings of the sixth conference on Computer systems, 2011.
[32] Amir Erfan Eshratifar, M. S. Abrishami, and M. Pedram, JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services, arXiv preprint arXiv:1801.08618, 2018.
[33] Mahdi Nazemi, Amir Erfan Eshratifar, and Massoud Pedram. 2018. A Hardware-Friendly Algorithm for Scalable Training and Deployment of Dimensionality Reduction Models on FPGA. In Proceedings of the 19th IEEE International Symposium on Quality Electronic Design.
[34] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, Huazhong Yang, A Survey of FPGA Based Neural Network Accelerator, arXiv preprint arXiv:1712.08934, 2018.