Project Plan: Deep Learning on Mobile Devices

JI ZHUORAN, 3035139915

I. Introduction

In recent years, we have witnessed the great success of General Purpose Graphics Processor Unit (GPGPU) in massively computing task. This achievement encourages processor manufactures to improve general computing capability of Desktop GPUs. Nowadays, programmable GPUs are also available on mobile devices, such as smartphones, autopilot cars, and IoT devices, which lead to significant performance boost and substantial energy reduction for massively mobile computation tasks [1].

Deep learning also drew significant attention recent years, especially in computer vision [2], speech recognition [3] and natural language processing [4]. Almost all of recent successful systems in these areas are built based on the deep neural network. However, even these technologies are critical to many mobile-phone apps, only a few of them take advantage of deep learning techniques [5]. This situation is caused by limited computation ability and memory space on mobile devices. Additionally, as the network grows more complicated, computation is also increasing exponentially, that makes deployment even more intractable [6].

Mainstream of these successful attempts of mobile deep learning usage is based on cloud [5], which has a lot of drawbacks, such as affecting privacy confidentiality, no real-time guarantee, and network overhead. However, porting deep learning framework to mobile or embedded devices is not trivial and is relatively under-studied especially on GPUs. There are a few of successful attempts in using mobile CPU for local execution, and CPUs present an attractive potential solution, especially because they are available on all mobile devices. However, CPUs will drain batteries in few hours if not few minutes, while most apps keep executing inference during running or even on background. As a result, CPU solution is not suitable for battery powered devices.

This project will introduce the MobileDL Toolkit, a deep learning toolkit that executed locally on mobile GPUs with reasonable speed and battery consumption. Instead of porting current framework directly, this toolkit is highly customized for mobile GPUs by taking computation, memory limitation and power consumption into consideration. Though our toolkit is customized for mobile GPUs, it is still universal as long as OpenCL is supported, as there is no assumption of specific GPU architectures. However, beyond code-level optimization, it offers two novel algorithms, namely: (1) Neural Network Minimization (NNM) and (2) converge points transition (CPT). Through these two algorithms, MobileDL automatically minimizes a convolutional neural network to balance the computation overhead and inference accuracy. Furthermore, a new fine-tuning scheme is introduced to transfer objective function into an equivalent one but is easier to minimize, by which MobileDL suppress error accumulated from multiple layers.

The following parts of this project plan has been organized as follows: first introducing the

fundamental terms related (§ II); next reviewing the literature based on two core concepts (§ III); then presenting two novel methodology for neural network minimization (§ IV); afterwards showing how MobileDL will be implemented(§ V) and finally giving a brief conclusion for this project plan (§ VI).

II. Preliminary

This section will begin with a primer on some basic concepts related to deep learning and mobile GPUs.

A. Mobile GPUs

Mobile GPUs have become increasingly powerful, which push forward the general computing technology for mobile devices over the past few years [7]. However, there are few papers discussing the general computing capability of mobile GPUs. Experience on desktop GPUs is not applicable on mobile GPUs, as design criteria of mobile GPU is different with desktop GPU. First of all, as mobile GPUs are usually powered by batteries, they are generally with lower frequency and much fewer cores [8]. Additionally, as most mobile GPUs are integrated into SoCs, graphics memories are not available [9][10] and accessing external memory will lead to much lower memory bandwidth. Last but not least, there are plenty of mobile GPU manufactures, such as Qualcomms Adreno family [9], Mali family [10], and NVIDIA Tegra family [11], leading to varies of mobile GPU architecture.

B. Convolutional Neural Networks

Convolutional Neural Networks (CNNs), which is composed of three major layers: convolution layers, pooling layers, and fully connected layers is the state-of-art neural network for vision and image related work [12].

The core operation in convolution layers is essentially 2-dimensional sliding-window convolutions with 2-dimensional convolution filters. Each convolution filter is related to one input channel and one output channel. It first convolves with its corresponding input activation plane and then accumulated to its output activation plane [13]. For a convolution layer with C input channels, K output channels, an $R \times S$ element filter is applied over a $W \times H$ element input channel to produce a $W \times H$ output activation plane. The overall time complexity is $\mathcal{O}(C \times K \times R \times S \times W \times H)$. Figure 1 shows a 2-dimensional matrix for illustration of a 4-dimensional tensor, and each small rectangle is a single convolution filter reshaped from $R \times S$ matrix to a vector.

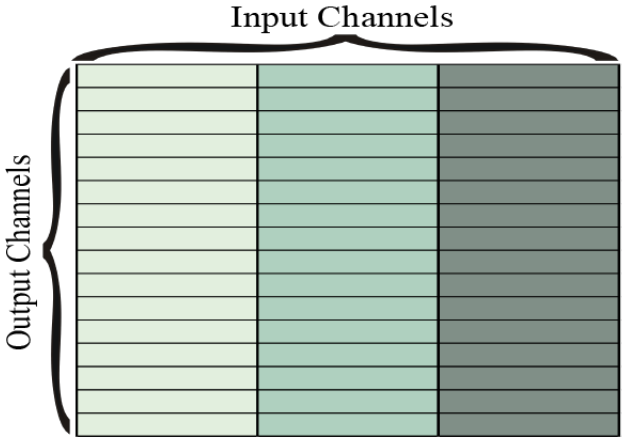

Fig. 1. Convolution Tensors: A convolution tensor with 3 input channels and 16 output channels, each small rectangle is a single convolution filter reshaped from $R \times S$ matrix to a vector.

For most CNNs, convolution layers dominate execution time of inference. For instance, for a typical neural network described in [14], on our platform: Snapdragon 820 development board, all its five convolution layers take 81.7% forwarding time, while all other layers only take the reminding 18.3%. The reason why convolution layers consume so much time is that, the order of time complexity of convolution layers is much higher than other layers. As discussed in the previous paragraph, the time complexity of convolution layers is $\mathcal{O}(C \times K \times R \times S \times W \times H)$, in other words, in each convolution layer, there are $C \times K$ convolution filters, and for each filter, $\mathcal{O}(R \times S \times W \times H)$ computation is needed, and $C \times K$ is usually more than 32×32 in state-of-art CNNs [14][15][16].

The time complexity of convolution layers make local execution intractable, however, Denil et al. has demonstrated that there are huge redundancies in neural networks [17]. They achieve an accurate prediction of all parameter within a layer by a small subset of them (about 5%), which implies neural networks could be heavily compressed. However, these redundancies are necessary during train as deeper neural networks provide larger capacity for functional approximation. Their work inspires us to apply K-means clustering method to explore the

redundancy in parameter space.

C. K-means clustering

K-means clustering is a method for vector quantization [18], originally from data mining, that is also popular for data clustering. The basic ideas of K-means clustering is partition n vectors into k clusters in which each vector is represented by the center of the whole cluster.

K-means clustering is a NP-hard problem [19]. However, many efficient heuristic algorithms converge quickly [20] [21], especially with initialization that is close to the final results. Because K-means clustering is applied in every iteration during training in our approach and parameters are updated smoothly in gradient descent method [22], the clustering result in the last iteration should close to clustering mean in this iteration. These two properties make it possible to use the previous result as initialization, which makes K-means clustering extremely efficient, so the overhead introduced by our approach is only minor.

III. Related Work

A. Deep Learning on Mobile Devices

Almost all popular deep learning framework, such as Caffe [2], Tensor-flow [23], Torch7 [24], Caffe2Go [25], Deeplearning4j [26], support Android platforms, and Shiro [27] took an important first step towards porting deep learning framework to mobile devices and achieve Cifar-10 recognition on Android devices in reasonable time. However, only a few of these frameworks adjust source codes for performance optimization on mobile devices. Even worse, all of them provide only CPU-based solutions for Android platforms, which are not feasible as discussed in section I.

DeepEye [28] demonstrated a device that is capable of executing several state-of-the-art deep vision models with nearly 17 hours battery life. DeepX [5] then proposed a decomposition method which split monolithic networks into unit-block of various types, significantly reduce latency of full connected layers. Both of these two works proved that notable speedup and power-saving could be achieved if mobile hardware-characteristics are taken into consideration when design the framework.

CNNdroid [29] proposed an Android GPU-accelerated library, which specifically designed and optimized for inference only on Android-Based mobile devices. Then DeepMon [30] showed early evidence that mobile device could handle large DNNs, and devised a suite of optimization techniques to reduce the processing latency.

Different with these previous works, MobileDL is a deep learning framework highly customized for less-powerful mobile devices. Also, MobileDL support OpenCL, which enables MobileDL running on heterogeneous SoCs, especially on power efficiency computation unit, such as GPU. Finally, different with CNNdroid and DeepMon, MobileDL is more aggressive, as little accuracy loss is permitted.

B. Neural Network Compression

After Denial et al. [17] proved the redundancies of neural networks, several CNN compression approaches have been proposed. Denton et al. [31] showed an early successful attempt of compressing the fully-connect layer by applying truncated singular value decomposition with insufficient of prediction accuracy. Then Gong et al. [6] exploited different vector quantization methods for neural network compression. Different with these previous works, which focus on reducing storage of network parameters, our approach focus on computation-reduction.

Jaderberg et al. [32] presented the speedup penitential of convolutional neural networks by low-rank decomposition of convolution tensors, they achieve about $2\times$ speedup on desktop CPUs. Then Lebedev et al. [33] demonstrated that using CP-decomposition on a full convolution tensor obtains $8.5\times$ CPU speedup with only minor accuracy drop (from 91% to 90%). Kim et al. [34] applied these compression techniques discussed above for fast and low-power mobile deep learning applications. They tested AlexNet [14], VGG-16 [15], and GoogleLeNet[16] in aspects of both energy consumption and execution time, and proved that these complexity state-of-art neural networks executing efficiently on mobile devices after compression. MobileDL extends these works by taking memory divergence and parallelism into consideration to make compressed neural networks efficiently execute on GPUs.

In the work that is most related to ours, Wu et al. [35] have proposed a unified framework for CNNs, named Quantized CNN(Q-CNN). Q-CNN quantize convolution tensors along the dimension of output channels. By splitting the weighting matrix into several sub-matrices and learning code-book on each of them, each sub-matrix is quantized into a smaller matrix with a code-book. Compressed outputs will be computed by convolution between smaller matrices and original inputs. Then desired outputs are reconstructed by searching compressed outputs and code-books. Our approach different with Wu et al.'s work in that our framework take advantage of massively computation unit such as GPU, so memory divergence introduced by Q-CNN should be removed. On the other hand, our approach does not introduce any memory storage overhead as no code-book nor sub-matrix need to be stored. Finally, our approach modifies fine-tuning method to drag the related convolution filters close to each other, by which accumulated errors are further reduced.

IV. Approach

Overall our approach is conceptually a simple two-phase methodology: neural network minimization during inference and converge points transition during training. In this section, first, we introduce an efficient test-phase convolution method with network minimization. Secondly, we demonstrate that better minimization can be achieved by fine-tuning the entire network using converge points transition.

A. NNM

For a convolution layer, its core operation is essential sliding window convolution. Convolution filters are stored as 4-dimensional tensors in CNNs, which can be denoted as $W \in R^{K\times C\times R\times S}$, where K and C are number of output channels and input channels, respectively. For each $W_{ij} \in R^{R\times S}$, it is the convolution filter corresponding to input channel i and output channel j, and $R \times S$ is the filter size.

In our approach, tensor is divided into c sub-tensor groups, each sub-tensor is denoted as $W \in R^{K\times R\times S}$, and a group is mathematically defined as $G_m = \{W_{ij}|\forall j = m\}$. Each G_m is then be treated as a set of vectors $S = \{v_1, v_2, ..., v_k\}$, where each vector v_i is a $R \times S$ real vector reshaped from W_{im}. For each S, NNM aims to partition these k vectors into $\tilde{k}$ sets by k-means clustering, then

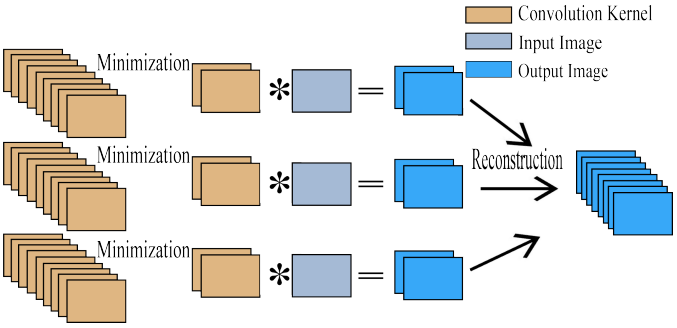

Fig. 2. Convolution Computation: For convolution filters refer to same input image, minimization is applied. Then, each input image is convolved with its corresponding minimized convolution filters to generate temporary outputs. Finally, actually outputs are reconstructed from these temporary outputs.

each set is represented by one single vector $\tilde{v}_k$, and the set of all representation vectors is denoted as $\tilde{S} = \{\tilde{v}_1, \tilde{v}_2, ..., \tilde{v}_{\tilde{k}}\}$. The mapping matrix is denoted as M, and $M_{ij} = 1$ if v_j is assigned to $\tilde{v}_i$, otherwise $M_{ij} = 0$. Then S' is reconstructed from $\tilde{S}$ and M, mathematically, $S' = \tilde{S}M$. The objective is to minimize "distance" between S and S'. Within-cluster sum of squares is chosen as loss function, mathematically, the follow objective function is going to be optimized:

$$min \sum_{i=1}^{n} \|v'_i - v_i\|^2$$

The computation process of convolution layers is illustrated in Figure 2, which has 3 input channels and 8 output channels. As the output dimensions are reduced from k to $\tilde{k}$, only $c \times \tilde{k}$ convolutions are actually computed, the temporary result is denoted as $O' = \left\{o'_1, o'_2, ..., o'_{\tilde{k}}\right\}$. Afterward, the original outputs can be approximately reconstructed from temporary $Output'$ and mapping matrix M, which can be mathematically expressed as

$$O = O'M$$

As a result, the overall time complexity can be reduced from $\mathcal{O}(C \times K \times R \times S \times W \times H)$ to $\mathcal{O}(C \times \tilde{K} \times R \times S \times W \times H + K \times W \times H)$. On the other hand, as only the clustered kernels and mapping index need to be stored, storage consumption can also be reduced.

B. CPT

With NNM, inference stage is significantly speedup. However, there is still a critical drawback:

the model which gives minimal before compression is not necessarily the one gives minimal after compression. As there are a lot of acceptable minimal of the objective function [22], a model which gives acceptable loss of the objective function before NNM is usually not the one give best classification accuracy after NNM. Furthermore, as NNM of each layer is independent of each other, the error will be accumulated. The accumulated error may be intolerable if the network is deep. With CPT, the parameters of the model are transited to other minimal of the objective function with least accuracy loss after NNM.

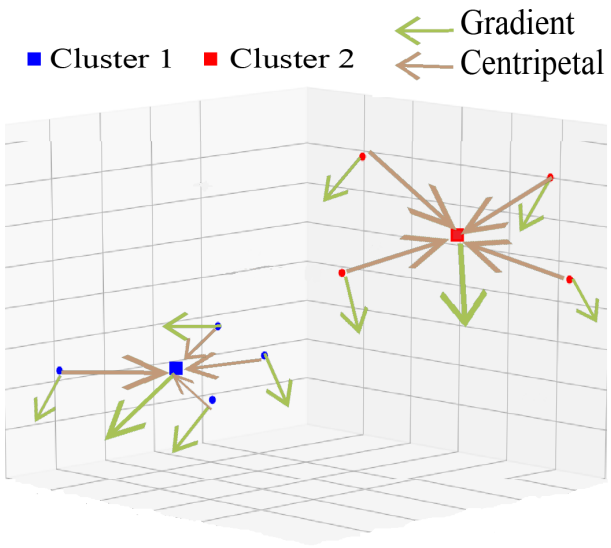

Fig. 3. CPT: There are two clusters, which are colored with blue and red, separately. Black arrow is centripetal direction, where green arrow is gradient direction. In a cluster, centripetal direction cancel each other, and the resultant direction is the sum of gradient direction.

CPT is essential a modified gradient descent method that adds a centripetal descent factor to the original descent direction. For state-of-art stochastic gradient descent, the parameters updating method is $\theta = \theta - \epsilon D$, where ϵ is the learning rate and D is the descent direction. Furthermore, the direction D is determined by two factor: gradient and regularization. The updating function then could be expressed as $\theta = \theta - \epsilon(g + \alpha\nabla\Omega(\theta))$, where g is the gradient, α is the weight decay rate, and $\nabla\Omega(\theta)$ is the regularization factor, for example, if L^2 regularization is used, then $\Omega(\theta) = \frac{1}{2}||\omega||_2^2$. This regularization strategy drives the weights closer to the origin, while CPT approach is based on driving the weights close to each other within same cluster to limit the various of model. The CPT factor is calculated as

$$\Psi(\theta) = reshape\left\{\tilde{S}_1 M_1, \tilde{S}_2 M_2, ..., \tilde{S}_c M_c\right\} - \theta$$

The new updating function now is

$$\theta = \theta - \epsilon(g + \alpha\nabla\Omega(\theta) + \frac{\beta}{\epsilon}\Psi(\theta))$$

This method can be intuitively illustrated in Figure 3. For simplification, each high-dimension vector is expressed as a point. Vectors assigned to the same cluster are in the same color. A cluster is represented by the mean of vectors within it and is expressed as a small square. To further simplify the problem, regularization factor is ignored. The descent direction of a vector could be treated as the combination of gradient direction and centripetal direction. If a cluster is regarded as a system, then all centripetal descent will cancel each other, as

$$\Sigma_i^k(\tilde{s}_i - v) = \Sigma_i^k \tilde{s}_i - kv = kv - kv = 0$$

Only the gradient descents contribute to resultant descent, which applied to the mean-vector, and the resultant descent is mathematically expressed as $D_s = G_s = \Sigma_i^k g_i$. Objective function will descent along the gradient in the granularity of cluster. Hence, after adding CPT factor, from system point of view, it is still the same optimization problem as before. Because within-cluster descent and between-cluster descent are independent, the whole parameter will also converge.

V. Project Methodology

The approaches discussed in the previous section will be implemented on Caffe [2]. There are two main reasons why we choose Caffe. First, as far as we know, Caffe is the only deep learning framework that supports OpenCL, and OpenCL is the only general computation API available on non-NVIDIA mobile devices. Secondly, we have already ported Caffe to mobile devices and gained state-of-art performance, and improving from the best solution is more convincing. Even though our approach is tested on Caffe with OpenCL, it is a universal approach that can be applied to any deep learning frameworks.

The implementation is composed of two main parts: (1) Modifying OpenCL code of convolution layer, (2) Adding centripetal descent in the solver. After implementation, we will test MobileDL on

several popular datasets and convolutional neural networks. We will train models with different minimization ratios. To make the training fast enough, we will use powerful desktops or workstations for training, while actually, training could be executed on mobile devices locally. Afterwards, both models and the toolkit will be deployed to our test platforms. Finally, speedup and accuracy loss will be measured and analyzed.

The detailed test plan is shown in Table 1. The neural network we test are AlexNet, VGG-16, and GoogleLeNet, and the datasets used are Mnist, Cifar-10, and LabelMe. Minimization ratio is chosen heuristically according to the capacity of the neural network.

TABLE I. DETAILED TEST PLAN

Neural Network	Dataset	Minimization Ratio
AlexNet	Mnist	[2, 4, 8]
AlexNet	Cifar-10	[2, 4]
AlexNet	LabelMe	[2, 4]
VGG-16	Mnist	[2, 4, 8]
VGG-16	Cifar-10	[2, 4, 8]
VGG-16	LabelMe	[2, 4, 8]
GoogleLeNet	Mnist	[2, 4, 8]
GoogleLeNet	Cifar-10	[2, 4, 8]
GoogleLeNet	LabelMe	[2, 4, 8]

VI. SCHEDULE

Tentative Schedule refers to Appendix.

VII. CONCLUSION

In this project plan, we propose a deep learning toolkit for mobile GPUs, and mainly focus on accelerating convolution layers. Extensive experiments will be conducted on state-of-art neural networks such as AlexNet [14], VGG-16 [15], and GoogleLeNet [16]. In expection, there will be up to $4\times$ speedup with only negligible accuracy loss.

REFERENCES

[1] K.-T. Cheng and Y.-C. Wang, "Using mobile gpu for general-purpose computing–a case study of face recognition on smartphones," in *VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on*. IEEE, 2011, pp. 1–4.

[2] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[4] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[5] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*. IEEE, 2016, pp. 1–12.

[6] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[7] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," in *Computer graphics forum*, vol. 26, no. 1. Wiley Online Library, 2007, pp. 80–113.

[8] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated cpu-gpu power management for 3d mobile games," in *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014, pp. 1–6.

[9] "Snapdragon 835 mobile platform with 10 nm 64-bit cpu," Aug 2017. [Online]. Available: https://www.qualcomm.com/products/snapdragon/processors/835

[10] Arm, "Mali gpu arm." [Online]. Available: https://www.arm.com/products/graphics-and-multimedia/mali-gpu

[11] "Nvidia tegra: The world's fastest mobile processors." [Online]. Available: http://www.nvidia.com/object/tegra.html

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[13] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," 2017.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

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

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[17] M. Denil, B. Shakibi, L. Dinh, N. de Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, 2013, pp. 2148–2156.

[18] R. Gray, "Vector quantization," *IEEE Assp Magazine*, vol. 1, no. 2, pp. 4–29, 1984.

[19] S. Arnborg and A. Proskurowski, "Linear time algorithms for np-hard problems restricted to partial k-trees," *Discrete applied mathematics*, vol. 23, no. 1, pp. 11–24, 1989.

[20] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko,

R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[21] K. Alsabti, S. Ranka, and V. Singh, "An efficient k-means clustering algorithm," 1997.

[22] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[24] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[25] "Delivering real-time ai in the palm of your hand." [Online]. Available: https://code.facebook.com/posts/196146247499076/delivering-real-time-ai-in-the-palm-of-your-hand/

[26] D. Team, "Deeplearning4j: Open-source distributed deep learning for the jvm," *Apache Software Foundation License*, vol. 2, 2016.

[27] sh1r0, "sh1r0/caffe-android-demo," Dec 2016. [Online]. Available: https://github.com/sh1r0/caffe-android-demo

[28] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," 2017.

[29] S. S. Latifi Oskouei, H. Golestani, M. Hashemi, and S. Ghiasi, "Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android," in *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 2016, pp. 1201–1205.

[30] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 82–95.

[31] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, 2014, pp. 1269–1277.

[32] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[33] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[34] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.

[35] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

TABLE II. TENTATIVE SCHEDULE

Index	Task	begin	end
1	Literature Review	01.09.17	01.10.17
2	project Plan	01.09.17	30.09.17
3	Read caffe code	03.10.17	10.10.17
4	Port Caffe to Mobile Devices	fnished in summer	
5	First Iteration for NNM	10.10.17	17.10.17
6	Second Iteration for NNM	18.10.17	25.10.17
7	Third Iteration for NNM	26.10.17	05.11.17
8	Milestone Test for NNM	06.11.17	10.11.17
9	First Iteration for CPT	11.11.17	18.11.17
10	Second Iteration for CPT	19.11.17	26.11.17
11	Third Iteration for CPT	27.10.17	05.12.17
12	Milestone Test for CPT	06.12.17	10.12.17
13	Detailed Intermediate Report	10.11.17	21.01.18
14	First Presentation	01.01.18	12.01.18
15	Port the Whole System to Embedded GPUs	22.01.18	27.01.18
16	Train Several Typical Models	28.01.18	02.02.18
17	Measure Inference Time on Mobile GPUs	30.01.18	04.02.18
18	Data Collection and Prepossessing	05.02.18	10.02.18
19	Final Report	11.02.18	11.04.18
20	Final Presentation	01.04.18	20.04.18
21	Project Exhibition	21.02.18	30.05.18