2018 Third ACM/IEEE Symposium on Edge Computing

# Enabling Deep Learning on IoT Edge: Approaches and Evaluation

Xuan Qi and Chen Liu
*Department of Electrical and Computer Engineering*
*Clarkson University*
*8 Clarkson Avenue, Potsdam, NY 13699, USA*
{*qix,cliu*}*@clarkson.edu*

***Abstract*—As we enter the Internet of Things (IoT) era, the size of mobile computing devices is largely reduced while their computing capability is dramatically improved. Meanwhile, machine learning technologies have been well developed and shown cutting edge performance in various tasks, leading to their wide adoption. As a result, moving machine learning, especially deep learning capability to the edge of the IoT is a trend happening today. But directly moving machine learning algorithms which originally run on PC platform is not feasible for IoT devices due to their relatively limited computing power. In this paper, we first reviewed several representative approaches for enabling deep learning on mobile/IoT devices. Then we evaluated the performance and impact of these methods on IoT platform equipped with integrated GPU and ARM processor. Our results show that we can enable the deep learning capability on the edge of the IoT if we apply these approaches in an efficient manner.**

***Keywords*-Machine Learning, Deep Learning, IoT, GPU, ARM, Quantization, Model Pruning**

## I. Introduction

Nowadays, machine learning (ML) technologies, especially deep learning (DL), have been widely applied in various fields and achieved state-of-the-art level performance. Meanwhile, mobile computing also keeps thriving and evolves to the Internet of Things (IoT). Consequently, the fusion of ML and mobile computing produces significant advancements in fields such as autonomous driving, health care, and identification technology. One step further, people also try to bring deep learning into IoT, especially at the edge of the IoT. If the IoT devices are only used as sensors or video stream capturing devices, the central cloud servers would need to sacrifice some processing power to handle low-level tasks such as video processing, data fusion, face or object detection. As a result, one promising approach is to move these low-level processing to edge devices, so as to free central servers from these tasks and let them perform more high-level processing such as pattern extraction and content analysis, etc.

But many of the deep learning models come with a huge amount of parameters, which require a very high computing overhead. Hence, porting deep learning model such as Deep Neural Network (DNN) onto IoT device is a challenging task. Because for most IoT devices such as surveillance cameras, communication devices and sensors, they are designed for low power with only limited computing capability, for example, few or a handful of ARM cores.

In this paper, we first conducted a brief review on the representative methods for enabling deep learning on mobile/IoT devices. Among all existing approaches, we mainly focus on evaluating three approaches: 1) Parallel acceleration; 2) Quantization; 3) Model pruning. In the experiment part, we evaluated the impacts of these methods on an exemplary Nvidia Tegra X2 platform, which is equipped with both ARM cores and integrated GPU.

## II. Related Works

In this section, we give a brief review of related researches on enabling deep learning on IoT edge devices. First, we review several typical approaches for ML model processing. What's more, since the DL models are another important factor in this research area, we also review the approaches for model optimization towards the IoT environment.

### A. Related works for IoT

For running the ML models in IoT, people have done works by the following configurations:

*1) Direct ML model processing on mobile/IoT-edge devices:* Cheng et al. [2] performed face recognition (FR) on mobile phone equipped with integrated GPU. In their experiment, the time for feature extraction and recognition for single face image is at seconds level, which is not suitable for processing real-time video sequence. Mandal et al. proposed an FR solution on Google Glass with ARM processor inside [7], which is similar to surveillance cameras equipped with only ARM cores. In their work, the video sequence is limited to $640 \times 360$ in size, which indicates the computation overhead will increase dramatically if we increase the resolution for different applications. From these works mentioned above, we can see that directly running ML models on IoT devices is still far from "satisfactory". Hence, more efficient framework is needed to handle ML computation in IoT.

*2) Accelerators for running ML model in IoT:* In order to accelerate the processing of ML models with the constraints of speed and latency, some researchers introduced middle-layer to offload some pre-processing works from the mobile devices on network edge. Tolga et al. [12] proposed Mobile

978-1-5386-9445-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SEC.2018.00047

Cloud Hybrid Architecture (MOCHA) to build a mobile-cloudlet-cloud framework, which applies GPU-accelerated cloudlet to transform face images to feature maps then transfers them to central server in the cloud for final processing of face recognition. With this middle-layer assisted framework, they successfully reduced the overall processing latency to meet the real-time requirement. Powers et al. [8] also applied cloudlet as middle-layer to handle the pre-processing stage of feature extraction for face recognition. Their experimental results showed that the overall processing frame rate can be accelerated by as much as $128X$.

### B. Related works for improving DL model

On the other side, researchers also investigated the potential of improving ML learning models.

*1) Execution Acceleration:* For the software side, we can get the ideal performance if the program can fully utilize the existing computing components. For example, Sun et al. [13] did a test on DL model execution by using ARM Compute Library (ACL), which is highly optimized for ARM platform. From the experimental result, we can see that the ACL-optimized version can reach a higher execution speed than non-optimized version.

*2) Model Quantization:* Another technique related to the hardware architecture is model quantization. For DL learning model itself, researchers already found that we can still reach almost the same level precision if we use model parameters, mostly the weights, with reduced precision [3], [6]. As a direct benefit, the memory overhead can be reduced by 50% or 75%, if we use 16-bit or 8-bit parameters instead of 32-bit, respectively. Moreover, with the cooperation of hardware, the quantization can also lead to faster model execution speed. For both GPU and ARM processor in IoT applications, a category of Single Instruction Multiple Data (SIMD) instructions is introduced to accelerate the add and multiply operations, which are the essential elements in DL model execution. For Nivida's GPU and ARM's Mali GPU, the model can be executed with 16-bit float or even 8-bit integer to get $2X$ to $4X$ speedup compared with 32-bit float. For ARM's latest processor with v8.2 instruction set, the ML models can also be executed in 8-bit or 16-bit mode.

*3) Model Pruning:* In additional to add more horsepower to IoT platform, people also tried to make the DL models themselves lighter with a similar level of performance. Han et al. proposed a model pruning method by only learning important connections [4]. With further optimization in quantization and weights sharing, they achieved higher than $10X$ compression ratio on model size and higher than $3X$ execution speedup. In another channel-based pruning method proposed by He at al. [5], they compressed the fully convolutional network of VGG model by $5X$ and the network with skip connections of ResNet by $2X$.

## III. Enabling Deep Learning Tasks on IoT edge

### A. Applied approaches in this work

As mentioned in Section II above, in order to enable DL tasks on IoT edge side, there are two main considerations, hardware acceleration and making DNN model lighter. As shown in Fig.1, we mainly focus on three approaches in this paper. For hardware acceleration, we consider multi-core execution and applying optimized instructions. For making DL model light-weight, we apply and evaluate the model pruning. Lastly, the quantization method is related to the optimization on both hardware and deep learning model levels.

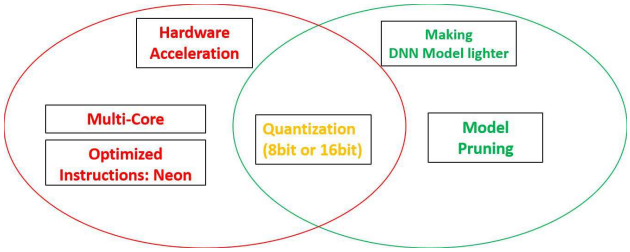


Figure 1: Methods for Enabling DL on ARM platform

### B. Technical enablers in this work

In this work, we use the ARM Compute Library (ACL) for the implementation on ARM platform and the Nvidia TensorRT for embedded GPU platform. Both frameworks not only provide functions for implementing ML models, but also come with low-level optimization towards hardware structures, which enables higher execution speed and efficiency.

*1) ARM Compute Library for ARM cores:* The ARM Compute Library (ACL) contains a comprehensive collection of software functions implemented and highly optimized instruction sets like NEON, an advanced SIMD extension set for the ARM Cortex-A family of CPU processors and the ARM Mali family of GPUs. The ACL library includes: 1) Functions for image and video processing like image re-size, edge detection, color conversion, among others; 2) Functions for building CNN models like: Activation layer, Convolution layer, Fully Connected Layer, Pooling Layer, among others.

*2) Nvidia TensorRT for Embedded GPU:* TensorRT is a light-weight framework provided by Nvidia with fine-grain hardware optimizing towards Nvidia GPU, such as the one on TX2 platform. Once we trained a DL model, we can convert the model structure definition and model parameters with the APIs or functions provided by TensorRT, and perform further quantization with reduced precision data type: 8-bit integer (INT8) or half-precision floating point (FP16). Compared with full length data type (FP32), the model with INT8 or FP16 can achieve $4X$ or $2X$ speedup on the platform which has the support of hardware optimization for executing operations with reduced precision data.

## IV. Experiments

In this section, we conduct our experiments on the Nvidia Jetson TX2 platform with one light-weight CNN model and one deep CNN model. First, both models are evaluated with different batch size and number of cores to show the effect of parallel acceleration on ARM cores. Second, we evaluate the effect and benefit of applying quantization on DL model. Third, we apply pruning on the VGG16 network and run the pruned model on ARM platform to verify the effect of using pruning on IoT edge devices.

### A. Experimental platform

The Nvidia Jetson TX2 platform used in this work is consisted of GPU and ARM cores. The GPU has 256 stream processor units (SPUs), which can provide enough computing power for tasks with high computing overhead and huge data bandwidth. Hence, the GPU is very suitable for running tasks with strict real-time demand. There are also six ARM cores with ARM-V8 instruction set support on TX2 platform. Among these 6 cores, two of them are high performance "Denver" cores custom-designed by Nvidia, which are suitable for handling Operating System (OS) tasks and main programs; the other four cores are power-efficient ARM A57 cores, which are suitable for handling non-urgent tasks or tasks that can be accelerated with higher parallel degree.

### B. DNN model for evaluation

For IoT edge devices, people intend to implement light-weight ML models like CNN models with few convolution layers. The consideration behind this choice is the trade-off among accuracy, latency and relatively limited computing capability provided by IoT edge devices. Besides, in those cases without a strict latency demand, people also run larger-scale DNN like VGG-16 with a deep structure on the IoT devices. Hence, in our experiment, we evaluate two DL models, one light-weight model and one larger DL model.

*1) Light-Weight DL model - CNN for frame selection from real-time video sequence:* Light-weight DL models are widely applied for the pre-processing of voice/video sequence which has real-time demand due to its acceptable computing overhead. In the experiment, we use a CNN model for Face Quality Assessment (FQA) in video as an example. The function for FQA model is to evaluate face image quality and pick the best frame for the following processing and reduce the data volume [9], [10]. The CNN architecture of our FQA module is shown in Table I. Overall, the light-weight CNN has four blocks of convolution operation, three of which are ordinary convolutional operations. The one special block called inception module [14] is a block which contains four parallel paths with different convolution kernel and has a combined output of all paths. The last sigmoid function is used to remap numeric prediction value within the range between $0.0$ and $1.0$.

Table I: CNN architecture in our FQA module

| Block Name | Kernel size | Num. of Feature Maps |
|---|---|---|
| Input | $64 \times 64$ | 1 |
| CONV1 | $3 \times 3$ | 12 |
| Inception | 4 paths: $1 \times 1$, $3 \times 3$, $5 \times 5$ convolution and max pooling | |
| CONV2 | $3 \times 3$ | 96 |
| CONV3 | $3 \times 3$ | 128 |
| FC1 | - | 128 |
| FC2 | - | 64 |
| Output (Sigmoid) | - | 1 |

Table II: Structure of VGG16 model

| | Num. of Layers | Feature Map Size | Num. of Conv. Kernels |
|---|---|---|---|
| Conv. Block 1 | 2 | $224 \times 224$ | 64 |
| Conv. Block 2 | 2 | $112 \times 112$ | 128 |
| Conv. Block 3 | 3 | $56 \times 56$ | 256 |
| Conv. Block 4 | 3 | $28 \times 28$ | 512 |
| Conv. Block 5 | 3 | $14 \times 14$ | 512 |
| FC layers | 3 | - | - |

*2) Large DNN model - VGG16:* We also use VGG16 [11], a DL model which has 16 convolutional and fully connected layers (except pooling layers) in total for evaluating the optimization methods towards the IoT platform. The structure of VGG16 is shown in Table II. The VGG16 model groups several convolutional layers into one block and processes the same size of feature maps within the block. The model applies $3 \times 3$ as the unique convolutional kernel and $2 \times 2$ kernel in all max-pooling layers. From the table we can see that VGG16 is a much bigger model than the FQA engine described in Table I according to its depth and feature map sizes.

### C. Implementing DNN model on ARM platform

For running DL models on ARM cores, we evaluate both small-scale model like our FQA CNN and large scaled model like VGG-16[11] which is widely used for various purposes.

*1) Light-weight CNN:* We first run the light-weight CNN model on four ARM cores. From the experimental results in Table III and Fig.2, with more cores, we can run the DL model in a parallel way and get a lot of speedup. One key factor we need to mention here is the processor occupancy. As we can see, if we only forward one image at one time, the multiple cores are not fully utilized and the speedup is not increasing obviously if we add cores from 2 to 4. Hence, for small scale DL model, a larger batch size with more images being forwarded to the model at same time will lead to a higher processor utilization and a higher speedup. From

Fig.2 we can observe the speedup trend of batch size 4 is better than batch size 1.

Table III: Running light-weight DL model on ARM

| Batch Size | Num. of cores | Total Time(s) | Throughput (faces/sec) | speedup |
|---|---|---|---|---|
| 1 | 1 | 51.073 | 52.4 | - |
| | 2 | 36.580 | 73.1 | 1.396 |
| | 4 | 33.326 | 80.2 | 1.533 |
| 2 | 1 | 51.316 | 52.1 | - |
| | 2 | 34.215 | 78.1 | 1.500 |
| | 4 | 23.432 | 114.1 | 2.190 |
| 4 | 1 | 51.931 | 51.5 | - |
| | 2 | 30.373 | 88.0 | 1.710 |
| | 4 | 20.957 | 127.5 | 2.478 |

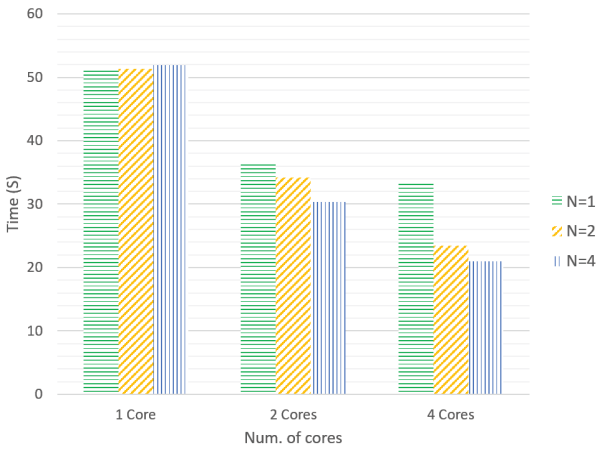


Figure 2: Comparison with different batch size and core number for light-weight CNN

*2) Larger DL model - VGG16:* We then run a larger DL model, the VGG-16 on four ARM cores. As we can see from Table IV and Fig. 3, for large scale DL model, the processor is fully occupied when the batch is only 1. And the speedup is only related to the parallelism degree, which is the core number.

### D. Quantization

*1) Running quantized DL model on embedded GPU:* As described in Section IV-C, we use the FQA CNN as an evaluation target to analysis the effects of quantization. To evaluate the processing speed, we test with videos took under two different scenes with three different resolutions: $1920 \times 1080$, $1280 \times 720$, and $640 \times 480$. The results of average frame rates across different scenes and different resolutions are shown in Table V. For here, the baseline implementation means running the CNN model with full length data type FP32, which is the un-quantized version. For the quantized version, we quantize the CNN model parameter to 16-bit float with TensorRT framework, which

Table IV: Running VGG16 DL model on ARM

| Batch size | Num. of cores | Total Time (s) | speedup |
|---|---|---|---|
| 1 | 1 | 6.10 | - |
| | 2 | 3.52 | 1.73 |
| | 4 | 2.48 | 2.46 |
| 2 | 1 | 12.09 | - |
| | 2 | 6.525 | 1.85 |
| | 4 | 4.711 | 2.57 |
| 4 | 1 | 24.22 | - |
| | 2 | 13.274 | 1.82 |
| | 4 | 8.918 | 2.72 |

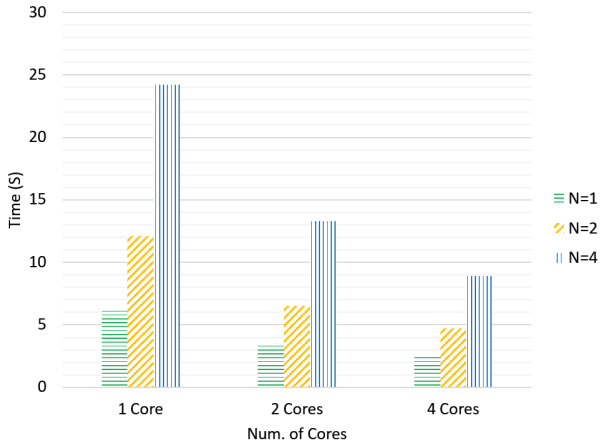


Figure 3: Comparison with different batch size and core number for VGG16

is the only supported reduced precision format on TX2 platform. In other word, the 8-bit wide integer format is not supported on this platform. From the result, we can see that for baseline implementation, it fails to reach the real-time speed (30fps) for all three resolution settings. Next, we show the results after quantization with 16-bit float point in the right column. After using quantization, we can successfully reach real-time processing speed (30fps) across all resolutions on TX2 platform. Especially for 1080p and 720p videos, we can see that we get 100% level performance gain after using quantized FQA CNN with half-precision float point weights.

The reasons for getting this satisfying speedup are: first, the TensorRT framework is highly optimized towards our platform's hardware architecture. Hence, the overhead of running deep learning support framework itself is reduced. Second, the accumulative computing overhead brought by this light-weight CNN model is reduced since we use 16-bit precision instead of 32-bit. In other word, the 32-bit wide computing unit in GPU can perform two 16-bit operations at the same time, which means a 100% theoretical speedup compared with performing 32-bit operations for models

Table V: Processing speed comparison before and after using quantization

| Video Resolution | Average Processing Speed (fps) | |
|---|---|---|
| | Baseline version (FP32 data format) | Quantized version (FP16 data format) |
| 1920 × 1080 | 16.17 | 34.25 |
| 1280 × 720 | 16.42 | 34.64 |
| 640 × 480 | 27.3 | 37.23 |

without quantization.

*2) Accuracy impact analysis of using Qantization:* We also perform a calibration procedure after model quantization. The calibration is a process which adjusts the output as close as possible to original output for quantized DL model. During the calibration, the output of quantized model is compared with original output and the parameters are tuned according to the difference, and the tuning target is to minimize the difference. We choose the FQA CNN model as the calibration target due to its regression nature, whose output is continuous values but not discrete class number of classification task. So it is a more difficult task for calibration and a good example for showing the calibration effects. Among all 2673 images in original test-set, we test with 2573 images, and only use the other 100 images for calibration. The result shows that the quantization only causes 1.005% average difference with the original FP32 version, which means the accuracy is kept so well while having dramatic speedup after applying quantization.

*3) Quantize large DL model:* We also test the VGG16 network with 8-bit integer quantization on discrete GPU. We compare the inference performance between FP32 and INT8 version of VGG-16 model. The dataset we use is the test-set in VGGFace2 dataset [1]. In Fig.4, we can see a $3.48X$ speedup under 128 batch size. And we can also get similar conclusion as Section IV-C, which is larger batch size can increase the hardware occupancy and get higher speedup.

### *E. Model pruning on ARM platform*

Following the approach described in [5], we list the computation overhead of each layer before and after pruning in VGG-16 model in Table VI. The computing overhead is calculated based on Equation 1, which is the total number of MACs (Multiply-Accumulates). The $H_f$ and $W_f$ are the heights and width of input feature map. The $H_k$ and $W_k$ are the heights and width of covolutional kernel.

$$MACs = H_f \times W_f \times N_{in} \times N_{out} \times H_k \times W_k \quad (1)$$

The computing overhead is affected by both the number of input($N_{in}$) and the number of output($N_{out}$) kernels. We

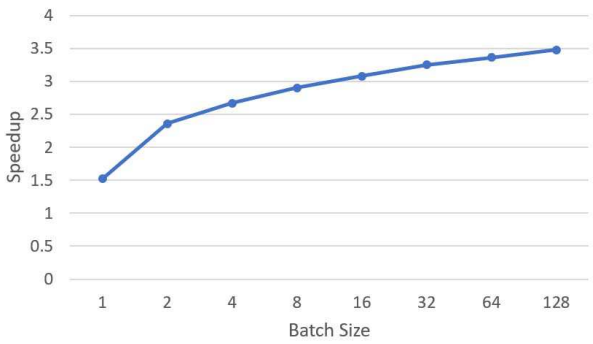


Figure 4: Speedup of using 8-bit Quantization under different batch size

can observe that pruning not only reduces the number of convolutional kernels on current compute stage, but also leads to a smaller input size for the next stage. As we can see, the theoretical speedup of using pruning is around $4.54X$ in total.

Table VI: Computing overhead comparison after pruning

| Layer Name | Feature Map Size | Number of Kernels | | GMACS | |
|---|---|---|---|---|---|
| | H*W | Before Pruning | After Pruning | Before Pruning | After Pruning |
| Conv1_1 | 224 × 224 | 64 | 24 | 0.087 | 0.033 |
| Conv1_2 | | 64 | 22 | 1.85 | 0.238 |
| Conv2_1 | 112 × 112 | 128 | 41 | 0.925 | 0.102 |
| Conv2_2 | | 128 | 51 | 1.85 | 0.236 |
| Conv3_1 | 56 × 56 | 256 | 108 | 0.925 | 0.155 |
| Conv3_2 | | 256 | 89 | 1.85 | 0.271 |
| Conv3_3 | | 256 | 111 | 1.85 | 0.279 |
| Conv4_1 | 28 × 28 | 512 | 184 | 0.925 | 0.144 |
| Conv4_2 | | 512 | 276 | 1.85 | 0.358 |
| Conv4_3 | | 512 | 228 | 1.855 | 0.444 |
| Conv5_1 | 14 × 14 | 512 | 512 | 0.462 | 0.206 |
| Conv5_2 | | 512 | 512 | 0.462 | 0.462 |
| Conv5_3 | | 512 | 512 | 0.462 | 0.462 |
| **Total** | - | - | | **15.38** | **3.39** |

In Table VI, we compare the execution time of VGG-16 model before and after pruning. The speedup of testing on ARM platform is close to theoretical analysis ratio of $4.54X$, which means the pruning is working effectively for real IoT application scenario.

Table VII: Execution time before/after pruning for VGG16 model

| Num. of cores | Execution Time (s) | | Speedup |
|---|---|---|---|
| | Before Pruning | After Pruning | |
| 1 | 6.10 | 1.43 | 4.27X |
| 2 | 3.52 | 0.83 | 4.24X |
| 4 | 2.48 | 0.58 | 4.28X |

## V. Conclusions

The advancement in both hardware and software research is making machine learning, especially deep learning possible on mobile devices and even IoT edge devices with limited computing capability. In this paper, we reviewed and evaluated three representative approaches, i.e., parallel acceleration, quantization, and model pruning, for IoT edge devices. We can discover optimization opportunities for enabling deep learning capability on IoT edge across hardware level to machine learning algorithm level. And multiple aspects should be considered together if we want to observe the optimization effects. For example, when we perform model quantization, not only transferring the model with reduced precision parameters need to be performed, but also the type of reduced precision computations supported by target hardware should be considered as well. In other word, in order to making deep learning on IoT edge plausible, a comprehensive consideration of hardware acceleration related approaches and deep learning model optimization approaches is a must.

## Acknowledgment

This material is based upon work supported by the Center for Identification Technology Research and the National Science Foundation under Grants No. 1068055 and 1650503. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan XP GPU used for this research.

## References

[1] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *International Conference on Automatic Face and Gesture Recognition*, 2018.

[2] 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*, pages 1–4. IEEE, 2011.

[3] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

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

[5] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, 2017.

[6] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N. E. Jerger, R. Urtasun, and A. Moshovos. Reduced-precision strategies for bounded memory in deep neural nets. *arXiv preprint arXiv:1511.05236*, 2015.

[7] B. Mandal, S.-C. Chia, L. Li, V. Chandrasekhar, C. Tan, and J.-H. Lim. A wearable face recognition system on google glass for assisting social interactions. In *Asian Conference on Computer Vision*, pages 419–433. Springer, 2014.

[8] N. Powers, A. Alling, K. Osolinsky, T. Soyata, M. Zhu, H. Wang, H. Ba, W. Heinzelman, J. Shi, and M. Kwon. The cloudlet accelerator: Bringing mobile-cloud face recognition into real-time. In *Globecom Workshops (GC Wkshps), 2015 IEEE*, pages 1–7. IEEE, 2015.

[9] X. Qi and C. Liu. Gpu-accelerated key frame analysis for face detection in video. In *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*, pages 600–605. IEEE, 2015.

[10] X. Qi, C. Liu, and S. Schuckers. Boosting face in video recognition via cnn based key frame extraction. In *The 11th IAPR International Conference on Biometrics (ICB), 2018 IEEE*. IEEE, 2018.

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

[12] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000059–000066. IEEE, 2012.

[13] D. Sun, S. Liu, and J.-L. Gaudiot. Enabling embedded inference engine with arm compute library: A case study. *arXiv preprint arXiv:1704.03751*, 2017.

[14] 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*, pages 1–9, 2015.