Deep Learning on Mobile Devices with Neural Processing Units

Tianxiang Tan
Pennsylvania State University

Guohong Cao
Pennsylvania State University

***Abstract*—Neural Processing Units (NPUs) have been developed for accelerating deep learning on mobile devices. Although the processing time of running DNNs on NPU can be substantially reduced compared to CPU, its accuracy becomes lower. To address this problem, we point out three research directions, i.e., model retraining, model partition, and computation offloading, to improve the performance of running DNNs on mobile devices with NPU.**

Deep Nerual Networks (DNNs) have enabled many emerging artificial intelligence applications such as augmented reality, virtual reality, google translate, etc. and there is a tremendous demand for running these applications on mobile devices. However, DNNs are computationally intensive, which creates many technical challenges for running DNNs on battery-powered mobile devices.

To address these challenges, many companies such as Qualcomm, Huawei, and Samsung have developed AI accelerators called Neural Processing Units (NPU). Different from traditional CPU which is good at processing serialized instructions, NPU is designed to perform better for parallel computations such as DNNs which involve millions or even billions floating-point computations. As a result, NPU can substantially reduce the processing time of deep learning on mobile devices. For example, on Mate 10 pro, the running time of VGG model can be reduced from 2600ms to 120ms by using NPU instead of CPU. Moreover, the power consumption of NPU is only 500mW which is about 83% lower than that of CPU (3W).

However, NPU has some fundamental limitations. First, different from CPU which uses 32-bit floating-point numbers for computation and storage, NPU uses 16-bit or 8-bit. Second, only a limited number of operations are supported by NPU, and hence some operations required in deep learning models have to be approximated. Finally, NPU has its own memory space, and the DNNs must be loaded into NPU to be executed. Since the memory space of NPU is small (about 200 MB in Mate 10 pro), most DNNs have to be compressed to run on NPU. Due to these limitations, the accuracy of running DNNs on NPU may be reduced.

Both processing time and accuracy are critical in many mobile applications. For example, a flying drone needs to accurately detect nearby obstacles in real time to avoid crashing, and many VR/AR applications need to interact with users through gestures and body posture recognition. NPU can accelerate deep learning, but it incurs accuracy loss. In this paper, to improve the performance of running DNNs on mobile devices with NPU, we identify the challenges, propose

Published by the IEEE Computer Society
© 2019 IEEE

solutions, and discuss future work along three research directions, i.e., model retraining, model partition, and computation offloading.

Understanding NPU

To have a better understanding of NPU, we conducted experiments with three kinds of phones: Pixel 6, Oneplus 7 and Mate 10 pro. Similar to [1], the experiments are based on four DNNs and their public data sets, VGG on the LFW dataset, VocNet [2] on the VOC dataset [3], ResNet-50 on the VOC dataset, and YOLO on the MS COCO dataset.

Figure 1 compares the processing time of running DNNs on CPU and NPU on different mobile devices. As shown in the figure, NPU is about 20 times faster than CPU on these three mobile devices. For example, to run Yolo on Pixel 6, NPU takes about 110ms while CPU takes about 4100ms. However, as shown in Figure 2, NPU suffers from accuracy loss and the loss varies based on the deep learning model. For instance, compared to CPU, using NPU on OnePlus 7 has similar accuracy when running VGG and ResNet, 30% accuracy loss when running VocNet, and 79% accuracy loss when running YOLO.

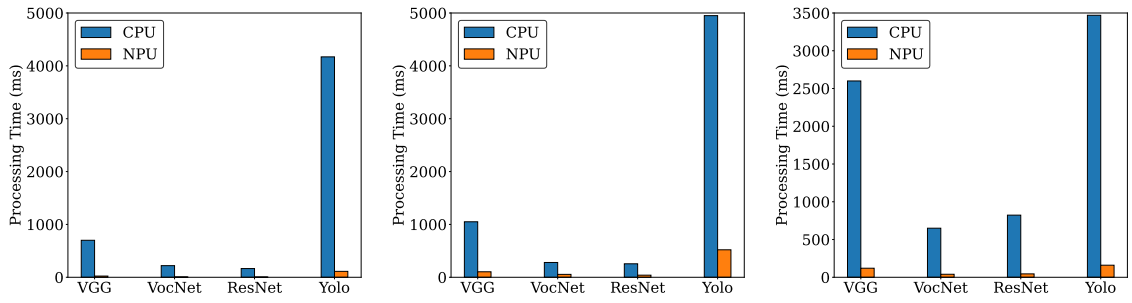
The accuracy loss is due to the limitations of NPU and it is also related to the complexity of the deep learning model. More specifically, VGG only compares the similarity between two feature vectors extracted from the face images. They are classified to the same person if the similarity is above a predefined threshold. Although NPU may introduce some small error to change values in the feature vector, the relationship between the similarity and the threshold does not change too much, and thus maintaining similar level of accuracy as CPU. However, YOLO is much more complex than VGG and it uses more information in the feature vectors to identify and locate multiple objects in the images. Each value in the feature vector represents the category, the location, or the size of an object. A small error introduced by NPU can change the prediction completely, and hence significantly affecting the accuracy.

Another observation is that the accuracy loss is different for different mobile device. The accuracy of running VGG on NPU with OnePlus 7 is about 40% higher than Pixel 6. This is because manufacturers design their NPUs differently, and use different toolkits to optimize the DNNs. As a result, the accuracy loss varies for different mobile devices.

Based on these experimental results, we observe that running DNNs on NPU may not always be the best option especially when accuracy is more important than processing time. To improve the performance of running DNNs on mobile devices with NPU, one research direction is to retrain the model with lower precision floating-point numbers since it is the most significant difference between NPU and CPU. Another research direction is to leverage model partition techniques to decompose DNN architecture into different layers running on heterogeneous processors, i.e., NPU is used to reduce the processing time of the computationally intensive layers while CPU is used for maintaining higher accuracy. These two research directions focus on accelerating DNNs by exploiting heterogeneous processors on mobile devices. The third research direction is to leverage computation offloading techniques to determine where to run DNNs based on the network condition, the special characteristics of NPU, and the optimization goal. In the following sections, we identify the challenges and propose solutions along these three research directions.

Model Retraining

The major difference between NPU and CPU is the precision of the floating point numbers. Advanced DNNs are trained and tested using 32-bit floating point numbers on powerful desktop CPU and GPU, and they are not designed for running on NPU. To address this issue, one solution is to retrain the DNNs, and there are a lot of research focusing on training DNNs with low precision numbers. For example, Wang *et al.* [4] proposed to use 8-bit floating-point numbers to train DNN models. Sun *et al.* [5] proposed gradient scaling and two-phase rounding techniques to minimize the quantization errors during training and they successfully used 4-bit floating-point numbers to train a DNN for classification. Yang *et al.* [6] modify the learning rate schedule to reduce the low-precision training time. However, these works focus on classification tasks with small images (i.e., using CIFAR-100 dataset). In contrast, there are other challenging tasks in the



(a) Pixel 6 (b) OnePlus 7 (c) Mate 10 pro

Figure 1. Processing time of running DNNs on CPU and NPU.

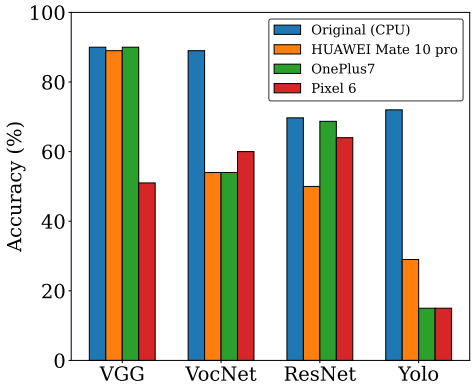
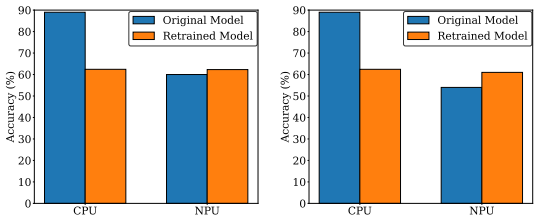


Figure 2. Accuracy comparison of running DNNs on CPU and NPU.



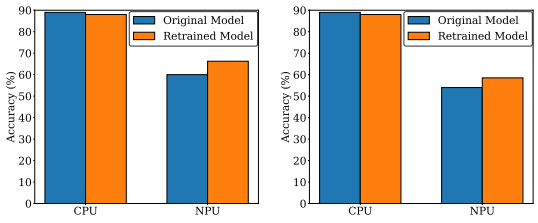
(a) Pixel 6 (b) OnePlus 7

Figure 3. Accuracy comparison of CPU and NPU using Float16-retrained VocNet.

real world such as face recognition and object detection which require more advanced DNNs, and these techniques can not be directly applied to retrain various advanced DNNs for NPU.

In this paper, we retrain VocNet using two different low-precision training strategies in the Tensorflow framework: Float16 and Mixed Float16. Float16 means that 16-bit floating point numbers are used in both computation and data storage during training. In mixed Float16, the computation is performed using 16-bit floating point numbers, but the data and parameters are stored using 32-bit.

Figure 3 shows the accuracy of the retrained VocNet using Float16. The figure does not show the processing time since the retrained model does not change the processing time. As can be seen from the figure, the accuracy of running retrained VocNet on CPU is about 28% lower than that of running the original model on CPU. This is because the VocNet is sensitive to the floating-point number precision. When the model converges, the loss of the retrained model is higher than the original model. Therefore, the accuracy of the retrained model is lower on CPU. Compared to the original model, the accuracy of the retrained model is higher on NPU. This is because Float16 emulates the floating-point number precision used on NPU, and the retrained VocNet is better adapted to NPU than the original model.



(a) Pixel 6 (b) OnePlus 7

Figure 4. Accuracy comparison of CPU and NPU using VocNet retrained with Mixed-Float16.

Figure 4 shows the accuracy of the retrained

VocNet using Mixed Float16. Compared to Figure 3, the retrained model can achieve similar accuracy as the original model on CPU. This is because Mixed Float16 uses 32-bit for data storage and the impact of numerical instability is much lower. As shown in the figure, the accuracy of running retrained VocNet on NPU is about 7% higher on Pixel 6 and 5% higher on OnePlus 7, but it is still much lower than that on CPU. This is because Mixed Float16 only performs computations using 16-bit floating-point numbers but store the data with 32-bit floating-point numbers. Different from CPU, NPU uses 16-bit floating-point numbers for both data storage and operations. Such difference may lead to the numerical instability on NPU and thus running the retrained model on NPU cannot achieve similar accuracy as that on CPU.

From the experiment, we can see that model retraining cannot improve the accuracy too much. Since NPU only supports 16-bit floating-point number or 8-bit integers, floating-point numbers may underflow or overflow when running some DNNs on NPU.

Model Partition

The basic idea of model partition is to decompose the DNN architecture into different layers running on heterogeneous processors, i.e., NPU is used to reduce the processing time of computationally intensive layers while CPU is used for maintaining higher accuracy. The model partition technique has been leveraged to reduce the computation time. For example, DeepX [7] divides the DNNs into different blocks which can be efficiently run on CPU or GPU. Neurosurgeon [8] optimizes energy by running the first few layers of the DNNs on local devices to reduce the data size and offloading the remaining part to the server for processing. Mao *et al.* [9] proposed to reduce the processing time by distributing the partitioned DNN models across mobile devices. Teerapittayanon *et al.* [10] proposed to reduce the processing time by adding early exit points to the DNNs, where a few layers are inserted into the DNNs to estimate the accuracy of the result. When the accuracy is above a certain threshold, the DNN execution will be stopped and the result will be returned. However, none of them considers the low accuracy problem introduced by NPU.

There are two important factors in model partition, accuracy loss and layer processing time. To measure them, we randomly selected 4000 images from the MS COCO dataset and run YOLO model for object detection using Mate 10 pro. We run one layer of Yolo on NPU while the remaining layers are run on CPU. As shown in Figure 5, running layer P_2 (a pooling layer) on NPU while other layers are run on CPU can reduce the processing time by 6% and incur 4% accuracy loss. Intuitively, a layer, for example C_6 (a convolutional layer), should be executed on NPU if the processing time can be substantially reduced with little or no accuracy loss. A layer (e.g., C_{17}) should be executed on CPU if executing it on NPU has much higher accuracy loss but little processing time reduction. However, most layers (e.g., C_{23}) are not in these two extreme cases, and hence it is hard to determine where to run them. When considering the overlapping effects of running multiple layers, the decision is harder. For instance, the accuracy loss of running C_6 and C_{22} on NPU while other layers on CPU is 0.08 which is not equivalent to the sum of their accuracy loss (i.e., 0.04). The accuracy loss of running the DNN with a layer combination depends on many factors, such as the number of additions and multiplications performed in each layer and the memory space occupied by the input/output data. Due to complex relationship between the accuracy loss and these factors, it is difficult to derive an equation to estimate the accuracy of running the DNN model with a layer combination.

For a DNN model with n layers, there are 2^n model partition decisions. For an advanced DNN model, there are usually dozens or even hundreds of layers and hence it is impossible to use brute force methods to find the best solution.

To address this problem, we propose heuristic based algorithms to find layer combinations which satisfy the application requirements on processing time and accuracy [11]. For example, in a flying drone application, detecting obstacles accurately with short processing time (in real time) is critical to avoid crashing. For these applications, maximizing the accuracy under some time constraint is more important, and we propose a Max-Accuracy algorithm to solve it. The basic

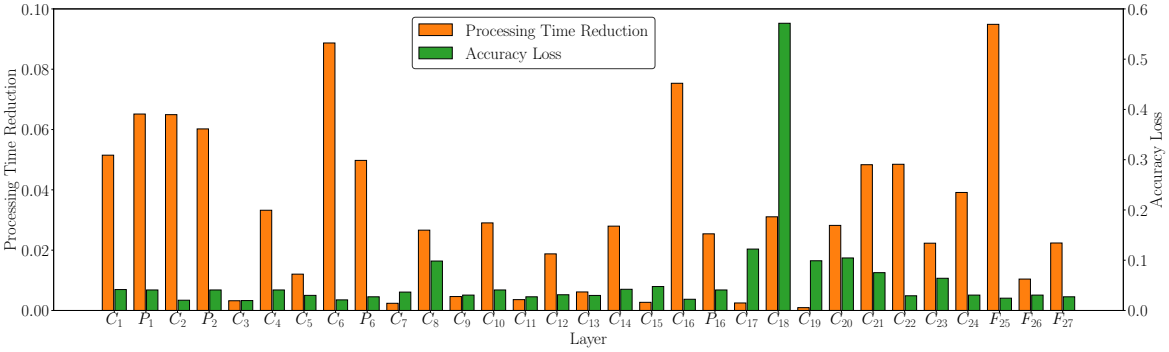


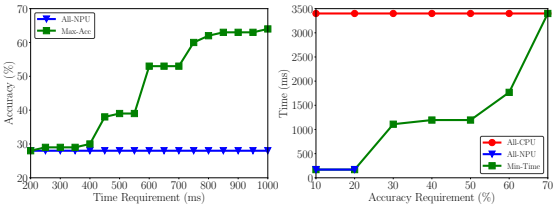
Figure 5. Processing time Reduction and accuracy loss when running one layer of Yolo on NPU while the remaining layers are run on CPU

idea is to move layers with higher accuracy loss from NPU to CPU as following.

(a) Initially, all layers are run on NPU.
(b) Sort them in descending order based on their accuracy loss.
(c) Starting from the first layer, move it from NPU to CPU until the processing time constraint is not satisfied.

For applications such as unlocking a smartphone, making payment through face recognition, accuracy is more important than the processing time. For these applications, we solve the problem of minimizing the processing time while ensuring the accuracy is above a certain threshold, by proposing a Min-Time Algorithm. The basic idea is to move layers with longer processing time from CPU to NPU as following.

(a) Initially, all layers are run on CPU.
(b) Sort them in the descending order based on their processing time.
(c) Starting from the first layer, move it from CPU to NPU until the accuracy requirement is not satisfied.

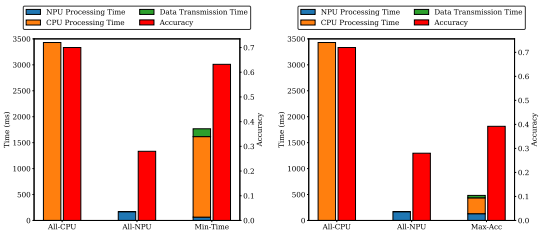


(a) Max Accuracy Algorithm (b) Min Time Algorithm

Figure 6. Performance comparisons for running YOLO on Mate 10 pro

To evaluate the performance of Max-Accuracy and Min-time, we compare them with All-CPU (i.e., the DNN model is always run on CPU) and All-NPU (i.e., the DNN model is always run on NPU). The experiment was conducted on HUAWEI Mate 10 pro, which is equipped with NPU. Its CPU is based on Octa-core (4 little cores and 4 big cores). The results are shown in Figure 6. In Figure 6(a), we did not draw the All-CPU approach since the processing time of running YOLO on CPU takes about 3.4s which is longer than the time requirement. As the time requirement increases, Max-Accuracy can improve the accuracy by moving more computations to CPU, but the All-NPU approach has the same accuracy. As shown in Figure 6(b), Min-Time outperforms All-NPU and All-CPU. Note that All-NPU can only reach the accuracy of 29%, and then it is not drawn after the accuracy requirement reaches 30%. Our proposed algorithms can significantly improve the performance compared to All-CPU and All-NPU by running computationally intensive layers on NPU to save time and running precision sensitive layers on CPU to maintain high accuracy.

Figure 7 shows the processing time of each approach. Since the proposed algorithm runs different parts of DNNs on heterogeneous processors, the data is moved between the main memory and the NPU memory. The data transmission time cannot be ignored because the NPU processing time is short and a large amount of data is transmitted between the main memory and NPU. As can be seen in the figure, the data transmission time occupies about 10% of the total processing time in Max-Accuracy and Min-Time. Since this



(a) Max Accuracy Problem (b) Min Time Problem

Figure 7. Details of the processing time comparisons for running YOLO on Mate 10 pro

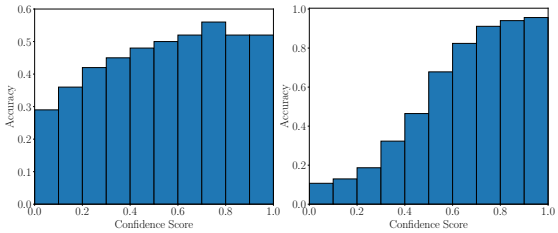
data transmission time is only a small fraction of the total processing time, it does not change the benefit of moving time-consuming layers from CPU to NPU. Thus, even considering this data transmission overhead, our proposed algorithms still outperform All-CPU and All-NPU as shown in Figure 6.

Computation Offloading

The aforementioned two techniques accelerate deep learning by exploiting heterogeneous processors on mobile devices. Such decision may be due to the limited network connection, the lack of server support, or privacy issues. For example, some users prefer processing sensitive data locally, and then many smart health applications belong to this category. For many other mobile apps such as augmented reality and cognitive assistance [12], users may be willing to achieve better performance by offloading the computation to the edge server [13]–[15].

Since the server has more computation capacity, more advanced deep learning models with high accuracy can be executed quickly. However, when the network condition is poor or when the data size is large which is usually true for video analytics, the offloading approach may take longer time because of the data transmission delay. On the other hand, the NPU based approach is faster, but with less accuracy. We propose an offloading framework [1] to combine these two approaches for real time video analytics on mobile devices, with the goal of maximizing accuracy under some time constraint.

In our offloading framework, the video frames are first processed on NPU which is very fast with negligible delay, and only the frames with low classification accuracy are offloaded to the server to improve accuracy. Thus, the key problem is to determine the classification accuracy, and we rely on the confidence score of running DNNs on NPU, which is computed based on the extracted feature vector of the DNN. If the confidence score is higher than a threshold, the classification result on NPU is most likely correct and can be directly used; otherwise, the data should be offloaded for further processing to improve accuracy.



(a) Original Confidence Score (b) Calibrated Confidence Score

Figure 8. Accuracy vs. confidence score.

The major challenge is that the confidence score of many advanced DNNs cannot accurately predict the classification results. To illustrate the problem, we conduct an experiment in which AlexNet is used to classify some video frames randomly selected from the FCVID dataset. The frames are divided into 10 bins with 10% confidence interval. For each bin, we compute the accuracy of running AlexNet on these frames. The result is shown in Figure 8(a).

Ideally, the confidence score and the classification accuracy should follow similar distribution. Then, the classification with confidence score 0.9 is more likely to be correct than that with a score of 0.5. However, as shown in Figure 8(a), the accuracy remains to be 0.5 for frames with confidence score much higher than 0.5 (e.g., 0.9). To address this problem, we have to calibrate the confidence score. Suppose the DNN model generates n different confidence scores $(x_1, x_2, \ldots, x_n)$, n logistic models will be trained for calibration, and each model i is used to calibrate a confidence score x_i. More specifically, model i takes the n confidence scores as input and outputs a new confidence score x_i. Figure 8(b) explains why calibrated confidence score is more effective. As the confidence score increases from 0 to 1, the accuracy increases from 0.11 to 1.

Offloading Scheduling

With the confidence score calibration, our framework determines which frames should be offloaded for further processing. Although there are some existing works [10], [16] on this topic, they relay on uncalibrated confidence score which is not effective for some DNNs, and a fixed confidence score threshold is used to determine which frames should be offloaded. However, the offloading decision depends on many factors, such as network bandwidth and the data size of the frames, and thus the fixed confidence score threshold may not work well.

To address this issue, we propose a confidence based offloading (CBO) algorithm. CBO adaptively adjusts the confidence score threshold based on the network condition, the data size of video frames and the DNN. In CBO, the frames with lower confidence score should be offloaded to increase accuracy if there is available bandwidth. CBO can also reduce the resolution of the offloaded frames to upload more frames with low confidence score. Suppose n frames have been processed locally on NPU. If a frame i with confidence score p_i is offloaded to the server in resolution r_i, CBO computes the offloading time T_i and the accuracy improvement $A(p_i, r_i)$ based on the frame data size, network condition and DNN accuracy information. Let $f(i,T)$ denote the maximum accuracy improvement by offloading the first i frames within time T. Our goal is to maximize $f(n,T)$. The overlapping subproblems can be represented as $f(i,T) = \max_{r_i}(f(i-1,T-T_i)+A(p_i,r_i))$. CBO solves the problem using a dynamic programming algorithm and selects a confidence score threshold to make offloading decisions for these n frames.

Figure 9 compares the performance of Local, Server, CBO and CBO-w/o under different network conditions. In the Local approach, the data is only processed on the local NPU and its performance does not change with the network condition. The Local approach has the same accuracy as CBO when the bandwidth is low since most frames have to be processed locally. However, as the bandwidth increases, CBO outperforms Local significantly because it can improve the accuracy by offloading the misclassified frames to the server for further processing. The server approach offloads all data to the server for processing. When the network condition is poor, its performance is much worse than CBO. This is because the frames have to be offloaded in an extremely low resolution to satisfy the time constraint, and running DNNs with low resolution frames cannot achieve high accuracy due to the information loss. The CBO-w/o is the same as CBO except that it uses uncalibrated confidence score to make offloading decisions. As can be seen, CBO outperforms CBO-w/o since the uncalibrated confidence score cannot accurately estimate the correctness of the classification result on NPU.

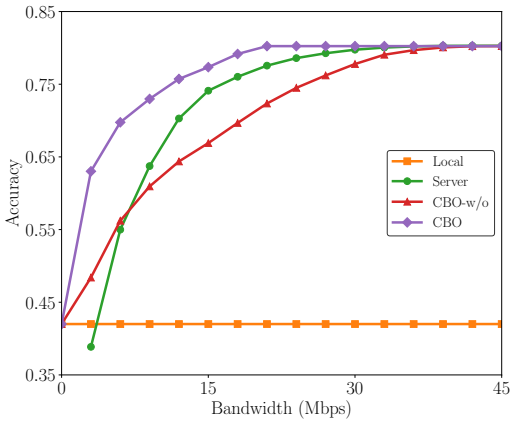


Figure 9. The performance of different approaches under different network conditions.

Discussions and Future Work

In this paper, we point out three research directions to accelerate deep learning on mobile devices with NPU. There are still many important issues for future research along these directions.

In model retraining, we leveraged the low-precision retraining method Tensorflow framework to retrain the VocNet. However, the result shows that the accuracy cannot be improved significantly after retraining. To address this issue, one possible research direction is to normalize the data and restrict the data range. For example, BatchNorm layers can be added before or after the activation layers to normalize the data. Moreover, since NPU only supports a limited set of operations, more research should be conducted to design DNNs or retrain existing DNNs based on these operations.

In model partition, we discussed how to decompose the DNN architecture into different lay-

ers running on CPU and NPU so that application requirements on accuracy and processing time could be satisfied. Although our proposed Max-Accuracy and Min-Time can outperform All-NPU and All-CPU, they only try a few layer combinations and select the best one based on heuristics. Since there are many layer combinations for a DNN, a better solution may exist. Since measuring the accuracy of a layer combination is time consuming, it is impractical to measure the accuracy for a large number of layer combinations. To efficiently search more layer combinations, the major challenge is to build a model to estimate the accuracy loss of a layer combination. This model is non-linear and depending on many factors, such as the accuracy loss of the layers, the number of layers running on NPU and the parameter size of the layers. As future research, machine learning techniques can be leveraged to estimate the accuracy loss and design algorithms to search for better layer combinations.

Another research direction is to consider the heterogeneity inside CPU such as the current big.LITTLE core architecture, where the big cores can provide better performance at the cost of more energy consumption, and little cores are slower with less power consumption. Considering various application requirements on processing time and energy, running all layers on either big core or little core may not be the best option. Therefore, instead of only using one type of core, we will study the performance and energy tradeoffs of big and little cores on running DNN models and optimize the model partition algorithms considering the big.LITTLE core architecture.

In computation offloading, we proposed a confidence based offloading algorithm to maximize the accuracy by determining which frames should be offloaded to the server for further processing. The current offloading framework only considers the tradeoff between processing time and accuracy. However, running computationally intensive DNNs on mobile devices also consumes a large amount of energy. Based on our experiment conducted on the HUAWEI Mate 10 pro, the power consumption of NPU is 500mW which is much less than the wireless interface. Offloading based approach can achieve higher accuracy, but it may cost more energy under poor network condition. On the other hand, NPU-based approach is energy efficient, but with less accuracy. Therefore, one future research direction is to study the tradeoffs among energy consumption, processing time, and accuracy in computation offloading.

CONCLUSIONS

In this paper, we identified the special characteristics of NPU, and proposed three techniques (i.e., model retraining, model partition, and computation offloading) to improve the performance of running DNNs on mobile devices with NPU. As the initial work on this new technology, we do not expect to solve all the problems. In the future, we will study how to retrain the DNNs based on NPU supported operations to further improve performance, to leverage machine learning techniques to search for better layer combinations in model partition, and to study the tradeoffs among energy, delay, and accuracy in computation offloading.

Acknowledgments

This work was supported in part by the National Science Foundation under grants CCF-2125208 and CCF-2215043.

REFERENCES

1. T. Tan and G. Cao. Deep Learning on Mobile Devices Through Neural Processing Units and Edge Computing. *IEEE INFOCOM*, 2022.
2. S. Lapuschkin, A. Binder, G. Montavon, K.-R. Müller, and W. Samek. Analyzing classifiers: Fisher vectors and deep neural networks. *IEEE CVPR*, 2016.
3. M. Everingham, S. Eslami, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes Challenge: A Retrospective. *Springer International Journal of Computer Vision (IJCV)*, 2015.
4. N. Wang, J. Choi, D. Brand, C. Chen, and K. Gopalakrishnan. Training Deep Neural Networks with 8-bit Floating Point Numbers. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
5. X. Sun, N. Wang, C. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. Maghraoui, V. Srinivasan, and K. Gopalakrishnan. Ultra-low precision 4-bit training of deep neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
6. G. Yang, T. Zhang, P. Kirichenko, J. Bai, A. G. Wilson, and C. De Sa. SWALP: Stochastic weight averaging in low precision training. *International Conference on Machine Learning (PMLR)*, 2019.

7. N. 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. *ACM IPSN*, 2016.
8. Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.
9. J. Mao, Z. Yang, W. Wen, C. Wu, L. Song, K. Nixon, X. Chen, H. Li, and Y. Chen. Mednn: A Distributed Mobile System with Enhanced Partition and Deployment for Large-Scale DNNs. *IEEE International Conference on Computer-Aided Design*, 2017.
10. S. Teerapittayanon, B. McDanel and H. Kung. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2017.
11. T. Tan and G. Cao. Efficient Execution of Deep Neural Networks on Mobile Devices with NPU. *ACM IPSN*, 2021.
12. Z. Chen, L. Jiang, W. Hu, K. Ha, B. Amos, P. Pillai, A. Hauptmann, and M. Satyanarayanan. Early Implementation Experience with Wearable Cognitive Assistance Applications. *ACM Workshop on Wearable Systems and Applications*, 2015.
13. E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making Smartphones Last Longer with Code Offload. *ACM Int'l Conf. on Mobile Systems Applications and Services (MobiSys)*, 2010.
14. M. Gordon, D. Jamshidi, S. Mahlke, Z. Mao, and X. Chen. COMET: Code Offload by Migrating Execution Transparently. *ACM USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.
15. L. Liu, H. Li, and M. Gruteser. Edge Assisted Real-Time Object Detection for Mobile Augmented Reality. *ACM International Conference on Mobile Computing and Networking*, 2019.
16. C. Hu, W. Bao, D. Wang and F. Liu. Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge. *IEEE Infocom*, 2019.

Tianxiang Tan received the BE degree from Sun Yat-sen University, the MS degree in computer science from University of Southern California, and the PhD degree in computer science from the Pennsylvania State University. His research interests include mobile cloud computing, edge computing and deep learning. He is a student member of the IEEE. Contact him at txt51@psu.edu.

Guohong Cao is a Distinguished Professor of computer science and engineering at the Pennsylvania State University. His research interests are mobile computing, wireless networks, machine learning, wireless security and privacy, and Internet of Things. Cao received a PhD in computer science from the Ohio State University. He is a Fellow of the AAAS and a Fellow of the IEEE. Contact him at gcao@cse.psu.edu.