2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW) | 978-1-6654-4989-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICMEW53276.2021.9456004

HARDWARE-AWARE MODEL OPTIMIZATION TOOL FOR EMBEDDED DEVICES

Cagri Ozcinar[1*], *Dongsung Kim*[1*], *Benjamin Rufus Duckworth*[1], *Shayan Joya*[1], *Nicolas Scotto Di Perto*[1], *Attila Dusnoki*[2], *Márkó Fabó*[2], *Dániel Vince*[2], *Gábor Lóki*[2], *Ákos Kiss*[2], *Christopher Alder*[1]

[1]Samsung Research United Kingdom, Staines-Upon-Thames, United Kingdom
[2]University of Szeged, Department of Software Engineering, Szeged, Hungary

ABSTRACT

Designing deep neural network models for embedded devices is a challenging task since the models need to be lightweight, fast, and accurate. This paper proposes a hardware-aware model optimization tool (HOT) to optimize a given model in terms of latency or accuracy by replacing its existing operators with the best-performing operators for target hardware. The proposed tool finds optimal operators with high accuracy and low latency in a short searching time while keeping the existing model structure rather than finding an entirely new network architecture. The result shows that the HOT improves MobileNetV2 backbone based models by up to 13.93% in accuracy (mAP) or 37.84% in latency (ms) with cascaded pyramid network (CPN) (pose estimation) and 31.03% in mAP or 56.64% in latency (ms) for single shot multi-box detector (SSD) (object detection) on digital television.

Index Terms— optimized model, embedded devices, hardware-aware, deep neural network

1. INTRODUCTION

On-device artificial intelligence (AI) has made significant progress in improving user experience. Recent embedded devices have been equipped with AI accelerators such as neural processing units (NPU) and tensor processing units (TPU) to improve the operation efficiency. Running deep neural network (DNN) models on these emerging embedded devices instead of resorting to cloud services becomes an effective solution for reducing costs, and obtaining low inference latency.

Embedded devices have limited computing power, so a DNN model designed to be executed in the cloud will not run due to device constraints, such as memory. The initial approach is to develop a suitable model for embedded devices manually. For instance, MobileNetV2 [1] uses depthwise-separable convolution rather than traditional convolution to reduce computational complexity. Recent works, however, show that some computationally efficient operations, such as depthwise-separable convolution, perform worse on TPU than a traditional convolution even though it is intended to reduce the number of parameters for less computation [2].

Current neural architecture search (NAS) techniques could present a suitable embedded DNN architecture by exploring a predefined search space but they require high computation cost due to their search algorithms [3]. For instance, MnasNet [3] reveals that a search for a suitable ImageNet classification model optimized for mobile phones, where each candidate is trained from scratch, can take 4.5 days on 64 TPU devices. The majority of NAS techniques have also ignored hardware constraints which are faced by many companies.

This paper proposes a practical hardware-aware model optimization tool (HOT) which has a full pipeline from model selection, hyper-parameter optimization (HPO) for the training to deployment on the target device. Our proposed optimization algorithm, called AutoSearch, quickly finds the most suitable DNN model for each target device under its constraints. To reduce the model search time, it uses a predefined search space and latency lookup table (LUT) created from target devices and utilizes integer linear programming (ILP) for the best model selection.

Our contribution in this paper is threefold. First, we propose a new model optimization tool that can find optimized DNN models which maximize the target device's utilization. Second, our algorithm reduces the architecture search time dramatically compared to the existing NAS techniques. Third, we explore the DNN optimization problem with two practical use-cases on commercial digital television (DTV) for the first time to the best of our knowledge.

2. RELATED WORKS

Designing optimized DNN models for embedded devices has been an active research topic in the recent years. Especially NAS has become increasingly popular for designing DNN networks optimized for embedded devices. MnasNet [3] was the first attempt to consider model latency as the main objective. It showed the output model is better than the human-designed ones. EfficientNet [4] goes further by using NAS to design a baseline model and create a family of models allowing scaling of the depth/width/resolution easily. Still, it does not consider the utilization of each target device for further

*Equal contribution

978-1-6654-4989-2/21/$31.00 ©2021 IEEE

improvement. Accelerator-aware NAS [2] highlights the benefits of customization of DNN architectures for the Google Edge TPU with building blocks for high overall utilization through reinforcement learning (RL) methods. However, an RL controller generally takes a longer time to find the best candidate model due to the sequential processing of the feedback. It has limitations to use a latency prediction simulator instead of the actual latency from the real hardware.

Our approach is to create a family of models by changing operators and scaling factors from the given baseline model. We maximize the target device's capacity to reduce the latency or improve accuracy by using the hardware's specific features. Instead of using expensive search algorithms (*e.g.,* RL and evolutionary computation), our proposed search algorithm, AutoSearch, finds the suitable candidate models by solving the formulated optimization problem using a LUT and a set of practical constraints.

3. MODEL OPTIMIZATION SYSTEM

HOT has model selection, training, and deployment modules as shown in Fig. 1. First, the model selection module performs a search with a given model by replacing operators and selecting the best tk candidate models. The AutoSearch algorithm utilizes a LUT built by measuring each operator's latency, floating point operations per second (FLOPS), and memory consumption on the target device and saving it in a database. Our algorithm determines a set of candidate models by solving the formulated ILP with practical constraints, such as latency budget and device memory capacity. Also, it could reduce the search time by training each candidate model in parallel with multiple GPUs.

Next, as the new model requires a new set of optimal hyper-parameters in training, the training module contains HPO. It supports grid search or Bayesian hyper-parameter optimization techniques [5], finding a set of optimal learning parameters (learning rate, optimizer, weight decay) for a given model.

Finally, the deployment module gives an interface to real target devices and invokes a hardware-level optimizer to quantize a given optimal model to 8 bit or 16-bit binary formats. It improves memory size and latency on DTV with tensor virtual machine [6, 7]. The deployment module provides a server capable of handling multiple requests for compilation and profiling, running on the real target device for generation of the LUT, and caching into the database for later usage as required.

4. MODEL SELECTION

In this subsections, we describe each part of the model selection module of the tool. In particular, we introduce our AutoSearch by formulating the MobileNetV2 model [8]. The HOT can equally be used by other models, such as ResNet [9], by creating their LUTs.

4.1. Search space

Search efficiency is an essential part of the practical NAS. For this reason, we make the search space discrete and constrain it with a set of candidate operators supported by the target device. The search is done at a block level in the given model, such as an inverted residual block (IRB) of the MobileNetV2.

We optimize MobileNetV2 by replacing its operators and applying variable expansion rates and kernel sizes throughout seven IRBs in Fig. 2. Our search space utilizes an expansion rate $\{3, 6, 9\}$ at IRBs 2-7, and a constant at IRB 1 $\{1\}$. Similar to the original design policy of the MobileNetV2, we use a constant expansion rate at the first block to avoid any unnecessary latency increase. We utilize kernel size $k \in \{3, 5\}$ as they are widely used in modern networks. We aim to search three different bottleneck structure types in each IRB: *a*) A bottleneck structure with depthwise pointwise convolutions in MobileNetV2 [8]; *b*) A bottleneck structure with standard 2D convolutions [2]; *c*) The inverse positioned pointwise convolutions with the sandglass block [10] as shown in Figure 2. Here, k denotes the selected kernel size.

4.2. Accuracy- and Latency-aware optimization

To obtain a set of accurate or low-latency models, S^*_{acc} or S^*_{lat}, for a given target device, each optimal model is chosen to maximize the model quality or to minimize the model latency performances. Based on the optimization objective, we determine the best tk candidate models for training. We first formulate the model selection problem using the following practical constraints:

1. **Latency:** A latency budget for a selected model to perform properly on a target device, L^{budget}.
2. **Memory:** A maximum memory limit for the selected model memory size on the target device, M^{max}.

In accuracy-aware optimization, our objective is to maximize utilization of a given target device and select the most optimal model for a given latency constraint. In this work, we do not take the power efficiency of the device into account. To reflect the model quality *without* a need for training, our formulation is based on the FLOPS count which is assumed to have a linear relationship with model quality [4, 11]. FLOPS is one of the most prevalent ways to estimate the amount of calculation in DNN model. In our own experiment, we have found a correlation coefficient value of 0.90 as a linear relationship between FLOPS and quality in terms of mAP for pose estimation task. We utilize the LUT which contains FLOPS cost, memory usage and inference latency measured on the target device for each possible IRB in MobileNetV2. Thus, our optimization problem can be formulated as follows:

$$S^*_{acc} : \arg\max_{c} \big(c^1_s x_s + \sum_{j \in \mathcal{J}} y_{ij} c^{2-7}_{ij}\big) \quad \forall i \in \mathcal{I} \quad \forall s \in \mathcal{S}, \ (1)$$

where c^1_s is the FLOPS cost for the s block type, $s \in \mathcal{S}$, and $\mathcal{S}$ is a set of different block types of IRB 1. Also, c^{2-7}_{ij}

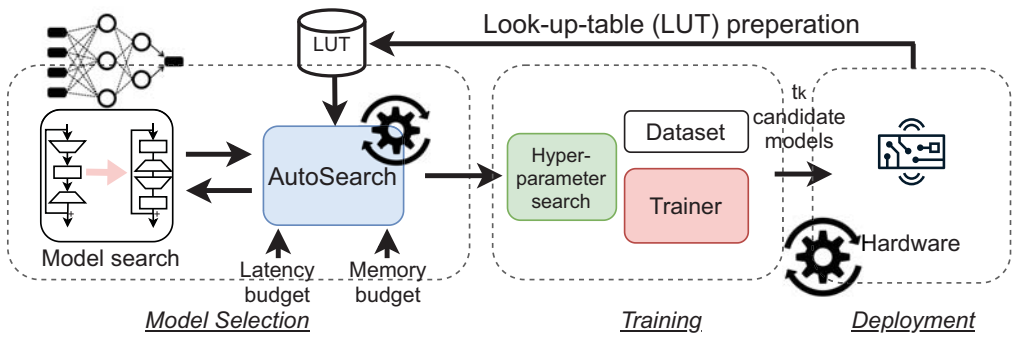

Fig. 1: Schematic of the proposed model optimization system.

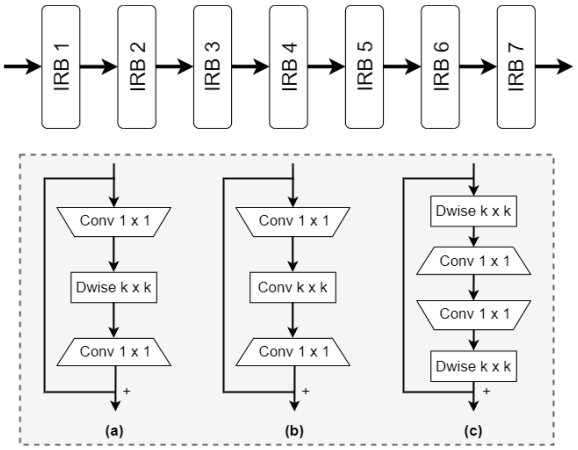

Fig. 2: Different types of blocks for MobileNetV2. (a) IRB with depthwise pointwise convolutions, (b) IRB with standard 2D convolutions, (c) The inverse positioned pointwise convolutions with the sandglass block.

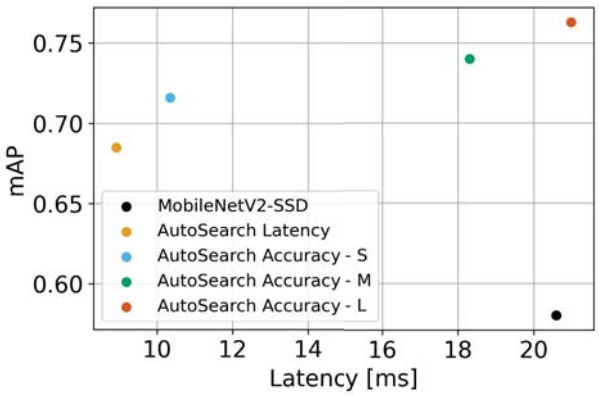

Fig. 3: Accuracy (mAP) vs NPU latency (ms) of the AutoSearch and the baseline (MobileNetV2 SSD).

represents FLOPS cost for the i block type of the j-th block, $i \in \mathcal{I}$. $\mathcal{I}$ is a set of different block types of IRBs 2-7. Here, x_s and y_{ij} are decision variables ($x_s = \{0,1\}$ and $y_{ij} = \{0,1\}$) for IRB 1 and for IRBs between 2 and 7, respectively.

Equation (1) maximizes the FLOPS cost and it is subject to the following constraints:

$$\sum_{s\in\mathcal{S}} x_s \leq 1 \quad \text{and} \quad \sum_{i\in\mathcal{I}} y_{ij} \leq 1 \quad \text{with} \quad \forall j \in \mathcal{J} \tag{2}$$

$$\sum_{s\in\mathcal{S}} l_s^1 x_s + \sum_{j\in\mathcal{J}} \sum_{i\in\mathcal{I}} l_{ij}^{2-7} y_{ij} \leq L^{budget}, \tag{3}$$

$$\sum_{s\in\mathcal{S}} m_s^1 x_s + \sum_{j\in\mathcal{J}} \sum_{i\in\mathcal{I}} m_{ij}^{2-7} y_{ij} \leq M^{max}, \tag{4}$$

where l_s and m_s are the inference latency and the device memory usage for the s block type, respectively.

Equation (3) addresses the **Latency** constraint by limiting the search space with a given latency budget, L^{budget}. Equation (4) satisfies the **Memory** constraint by setting the maximum allowed memory budget M^{max} of the feature extractor model, and $M^{budget} = M^{max} - M^{head}$. To estimate M^{budget}, we subtract the calculated memory for the operators outside of the feature extractor model, M^{head}, from the total memory capacity of a given embedded device, M^{max}. The ILP problem proposed in Eq. (1) can be solved by a generic solver GNU linear programming kit in less than one second on Intel(R) Core(TM) i9-9820X CPU @ 3.30GHz with 126 GB of RAM.

Our objective is to find a set of candidate models in *latency-aware optimization*, S^*_{lat}, which provides the lowest latency on a given target device. For this, similar to Bichen *et al.* [12], we estimate the total latency of the network by summing up the latency of each IRB in search spaces and determine the best tk candidate models.

5. EXPERIMENTS

We target an NPU device in a given DTV for model optimization and evaluate our tool on two different applications: SSD object detection [13] and CPN pose-estimation [14]. We defined tk as 8, which shows a good accuracy-latency trade-off with a reasonable search time. We train each candidate model on GeForce RTX 2080 Ti GPUs with 11GB memory. In our algorithm, AutoSearch Accuracy, we set L^{budget}=30 ms as the required maximum latency budget and defined three variants of models, small (S), medium (M), and large (L). We set 0.7, 1, and 1.3 as widths of MobileNetV2 to AutoSearch Accuracy -S, -M, and -L, respectively.

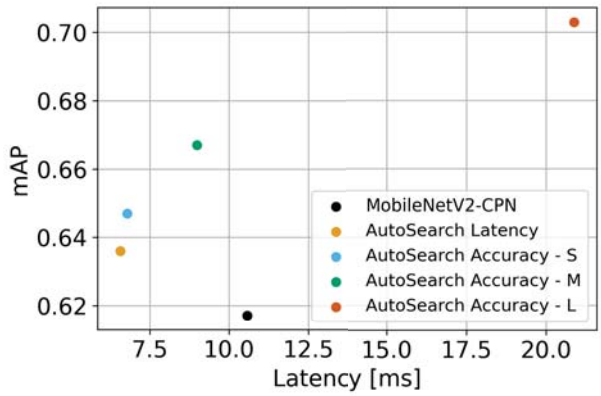

Fig. 4: Accuracy (mAP) vs NPU latency (ms) of the AutoSearch and the baseline (MobileNetV2 CPN).

5.1. Object detection

We merged Pascal VOC 2007 and 2012 datasets [15], and trained MobileNetV2 SSD for 400 epochs. We used ADAM optimizer, an initial learning rate with $\{0.0005, 0.0007\}$ and a factor of gamma $\{0.3, 0.5, 0.7\}$ with batch size 32. We used standard image augmentation techniques during training, including random crops, flipping, hue, saturation, contrast, and brightness distortions.

We compared our AutoSearch models with the original baseline MobileNetV2 [8] with SSD. Figure 3 shows the performance of accuracy versus NPU latency. Looking at the results, we observe that the AutoSearch latency model is 56.64% faster and has 23.79% better mAP than the baseline model. We also see that the proposed AutoSearch Accuracy model's small, medium, and large variants achieve 23.44%, 27.58%, and 31.03% mAP improvement on val2017 of COCO dataset [16], respectively.

5.2. Pose estimation

We used 2017 COCO multi-person keypoint dataset [16], and trained the network with ADAM optimizer and 32 batch for 90 epochs. We changed an initial learning rate with $\{0.005, 0.007\}$, a factor of gamma $\{0.3, 0.5, 0.7\}$ in a grid way. We used augmentation techniques such as zooming, flipping horizontally, rotating, and color dithering at training.

Similar to the previous experiment, we compared our models with an original baseline model, MobileNetV2 [8] with CPN. The results in Figure 4 show that the AutoSearch Latency model outperforms the baseline model in latency by 37.84% and accuracy by 3.25%. The small, medium, and large variants of the AutoSearch Accuracy model achieve 4.86%, 8.10%, and 13.93% mAP in val2017 of COCO dataset [16], respectively.

6. CONCLUSION

This paper presents an automated hardware-aware model optimization tool (HOT) for redesigning a given MobileNetV2 network using operators optimized for a target NPU device with minimum human intervention. Our approach has shown that it finds optimized models that maximize the target device's utilization to improve the latency or accuracy with a small amount of search time. We have demonstrated two applied vision tasks, object detection and pose estimation, to have better latency (ms) or quality (mAP) on a given digital television after the optimization process.

7. REFERENCES

[1] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[2] Suyog Gupta and Berkin Akin, "Accelerator-aware neural network design using automl," in *On-device Intelligence Workshop at MLSys*, 2020.

[3] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *CVPR*, 2019, pp. 2820–2828.

[4] Mingxing Tan and Quoc V Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv:1905.11946*, 2019.

[5] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy, "BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization," in *Advances in NIPS*, 2020.

[6] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al., "Tvm: An automated end-to-end optimizing compiler for deep learning," in *13th SENIX OSDI*, 2018, pp. 578–594.

[7] Sunwoong Joo, Attila Dusnoki, Martyn Bliss, Ben Duckworth, Nicolas Scotto Di Perto, Markó Fabó, Gábor Lóki, Dániel Vince, Ákos Kiss, and Cheul-hee Hahm, "A memory-aware performance optimization of tensor programs for embedded devices," in *ICCE-Asia*. IEEE, 2020, pp. 168–171.

[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018, pp. 4510–4520.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] Zhou Daquan, Qibin Hou, Yunpeng Chen, Jiashi Feng, and Shuicheng Yan, "Rethinking bottleneck structure for efficient mobile network design," in *ECCV*, 2020.

[11] Xuanyi Dong, Mingxing Tan, Adams Wei Yu, Daiyi Peng, Bogdan Gabrys, and Quoc V Le, "Autohas: Efficient hyperparameter and architecture search," *arXiv:2006.03656*, 2020.

[12] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *CVPR*, 2019, pp. 10734–10742.

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg, "Ssd: Single shot multibox detector," in *ECCV*. Springer, 2016, pp. 21–37.

[14] Yilun Chen, Zhicheng Wang, Yuxiang Peng, Zhiqiang Zhang, Gang Yu, and Jian Sun, "Cascaded pyramid network for multi-person pose estimation," in *CVPR*, 2018, pp. 7103–7112.

[15] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman, "The pascal visual object classes (voc) challenge," *IJCV*, vol. 88, no. 2, pp. 303–338, 2010.

[16] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick, "Microsoft coco: Common objects in context," in *ECCV*. Springer, 2014, pp. 740–755.