

Dynamic Neural Network Accelerator for Multispectral detection Based on FPGA

Xiaotian Wang^{*†}, Letian Zhao^{*†}, Wei Wu^{*†} and Xi Jin^{*†}

^{*}State Key Laboratory of Particle Detection and Electronics, University of Science and Technology of China, Hefei, China

[†]Institute of Microelectronics, Department of Physics, University of Science and Technology of China, Hefei, China

Emails: {wxtsdg, zhaolt, wuw1993}@mail.ustc.edu.cn, jinxi@ustc.edu.cn

Abstract—Multispectral detection is an important and challenging task in computer vision. Two-stream architecture is commonly used in multispectral detection, which requires much more computational resources, more on-chip memory, and higher off-chip bandwidth than traditional CNNs. Compared to static neural networks, dynamic networks have advantages in terms of computational efficiency, adaptiveness, etc. In this paper, we propose an efficient dynamic neural network for multispectral detection. Compared with the unoptimized static model, our proposed model reduces computation by 41% and off-chip memory access by 77% on average with less than 1% decrease in mAP. We design cross-layer scheduling for dynamic networks that can fine-tune the fusion model at runtime. We implement a flexible FPGA-based hardware accelerator to evaluate our design. Experiments demonstrate that our dynamic neural network accelerator achieves a maximum $2.7\times$ inference speedup.

Keywords—Multispectral detection, FPGA-based accelerator, Dynamic Neural Network, Corss-layer scheduling

I. INTRODUCTION

In recent years, significant breakthroughs in computer vision have been achieved with the development of computer technology. Real-world applications are often in complex and changing environments, such as rain, fog, and low light. In these cases, visible images can not provide enough information to achieve high accuracy. Thermal images can overcome these difficulties. Infrared thermal imaging technology has been developing and maturing. As a result, multispectral detection has been widely used in autonomous driving, security monitoring, search and rescue, and other fields [1]–[6].

In many application scenarios, such as autonomous driving, high accuracy and low latency are required for multispectral detection. At the same time, it is desired to reduce cost, chip area, and power consumption as much as possible. However, to ensure accuracy, multispectral detection commonly uses a two-stream architecture. Compared with the traditional CNN model, it requires more computational resources, more on-chip memory, and higher off-chip bandwidth. To address this problem, using dynamic neural networks is a feasible solution. Compared to static neural networks, dynamic networks selectively activate model components based on the input sample during inference. Studies [7] have shown that dynamic neural networks have greater computational efficiency and representation capability.

It is challenging to process dynamic neural networks efficiently, due to the significant irregularities introduced by the redundancy elimination at runtime. General hardware

accelerators are difficult to benefit from dynamic architectures, because they can not efficiently process optimized models, and eliminate redundant computing and memory access [8].

In this paper, we propose an efficient dynamic neural network for multispectral detection. We design an FPGA-based accelerator and the corresponding scheduling algorithm that can efficiently process dynamic neural networks. The main contributions of this work are summarized as follows:

- We design an efficient dynamic neural network for multispectral detection. Compared with the unoptimized static model, our proposed model reduces computation by 41% and off-chip memory access by 77% on average with less than 1% decrease in mAP value.
- We propose cross-layer scheduling for dynamic neural networks to improve the computation to communication (CTC) ratio. Furthermore, we designed a scheduling arbitration module, which can fine-tune fusion strategy according to the activation ratio of each layer at runtime to obtain the optimal scheduling scheme.
- We propose a flexible hardware architecture design for dynamic neural networks. Depending on the parameters of different convolutional layers and the activations at runtime, the accelerator can be adaptively deployed, significantly improving throughput and resource utilization.

II. BACKGROUND

A. Dynamic neural network

Dynamic neural network is an emerging research topic in deep learning, which can adapt the computational graphs and parameters during inference. In this paper, we mainly focus on sample-wise dynamic architectures. The network with dynamic architectures can adjust the depth, width, or perform dynamic routing according to different input samples. In this way, the redundant computation can be reduced when dealing with simple samples. When dealing with hard samples, it can preserve more representation power [7]. Compared with static networks, dynamic networks have higher flexibility and are widely used to process visible images in computer vision.

SkipNet [9] and convolutional network with adaptive inference graph (Conv-AIG) [10] implement dynamic layer skipping and adjust the depth of the network by introducing the gate module. Pooling operations and FC layers are used to generate channel-wise hard attention to make discrete decisions

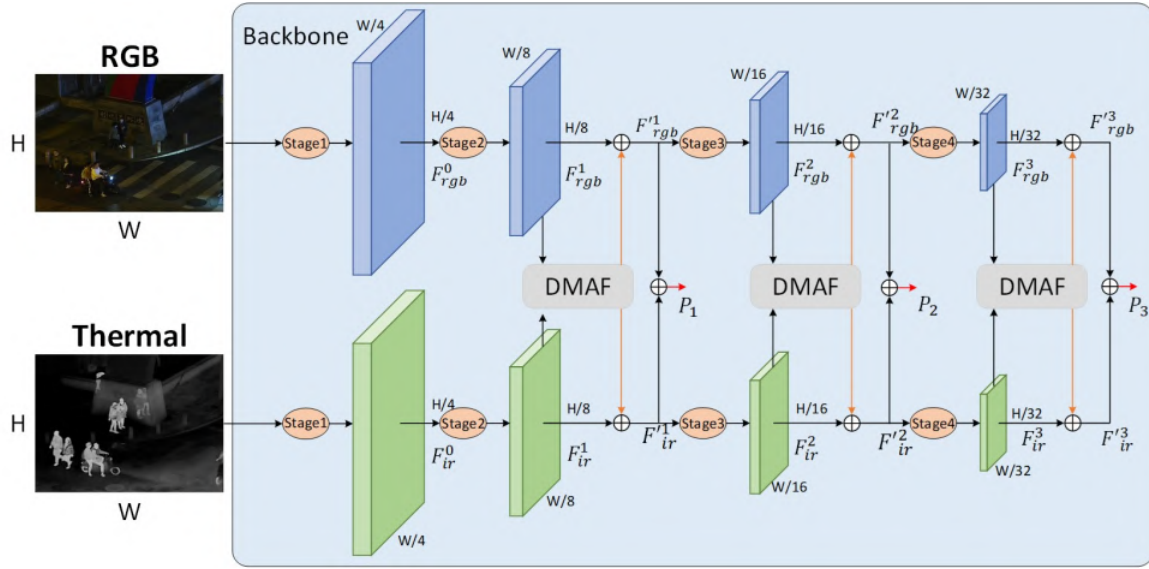


Fig. 1: Overview of multispectral detection Network's backbone.

on whether to activate each channel [11]–[14]. This kind of dynamic pruning with gating functions can adjust the network width adaptively. Compared to static pruning that permanently removes filters, this way improves inference efficiency without reducing model capacity [7].

B. Computation to communication ratio

Computing and communication are two key constraints limiting system throughput. Computation to communication ratio features the number of operations performed per memory access. A high CTC indicates that the layer is computationally intensive, and the inference latency is determined by the number of computational resources. On the contrary, a low CTC means the off-chip bandwidth greatly affects the performance [15]. According to the computing resources and bandwidth provided by the hardware, [16] draws the roofline model to explore the optimal design of the convolutional layer. [15] adopt the cross-layer scheduling, and further explore the optimal layer fusion strategy. Specifically, cross-layer scheduling is used to improve the CTC. While achieving the peak performance, the optimal design is obtained by the trade-off between on-chip memory and off-chip memory access.

Since [15] studies static networks, the design space exploration can be done offline, and the hardware only needs to load a configuration for deployment. However, for a dynamic neural network, the specific amount of computation and memory access can only be obtained at runtime, which brings difficulties to hardware optimization.

III. PROPOSED METHOD

A. Multispectral Detection Algorithm

We propose a dynamic neural network for multispectral detection, as shown in Fig.1. It is a two-stream model extended from Yolov5 [17]. The backbone is replaced by the lightweight MobileNetV2, in which the fusion module adopts Differential

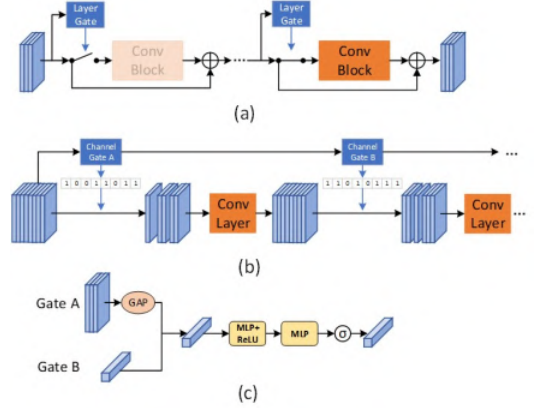


Fig. 2: (a) Layer gate; (b) Channel gate; (c) Gate A and Gate B

Modality Aware Fusion Module (DMAF) proposed by MBNet [18]. On this basis, we take two dynamic components: 1) Layer skipping; 2) Dynamic pruning, as shown in Fig.2.

Each stage contains multiple convolutional blocks, and the layer skipping module is embedded before each stage. The layer gate generates 1-bit data according to the input of the convolution block, and determines whether to skip the current convolution block. Dynamic pruning is used before convolutional layers. The Channel gate is embedded before the convolutional layer. According to the input, c-bits (c is the number of channels) of data are generated to determine whether each channel will be pruned. The layer gate and channel gate use Global Average Pooling (GAP) for dimensionality reduction. After two MLPs, the hard-sigmoid function is used to restrict the vector's range to (0,1). Finally, the vector is binary quantized to 0 or 1.

It is impractical for cross-layer scheduling if each channel

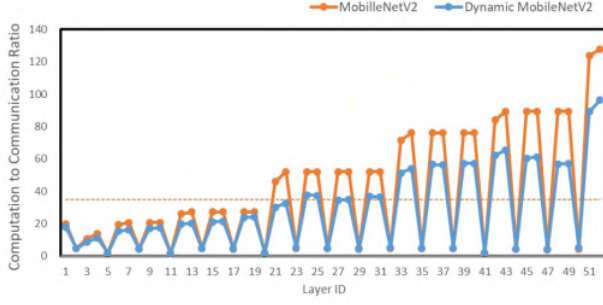


Fig. 3: The CTCs of static and dynamic neural networks.

gate module takes the feature of the current layer as input. GaterNet [19] builds an additional decision network to address this problem, but this brings greater computational overhead. So we take a mixed approach. Specifically, the input feature is sent to channel gate A, which is located before the first layer of one or several convolutional blocks. Gate B takes the output of the previous gate as its input. The network is divided into multiple subsets $\{set_0, set_1, \dots, set_i, \dots, set_n\}$ by channel gate A.

B. Cross-Layer Scheduling

For a specified FPGA, assume that the number of DSP slices provided by the FPGA is N , the utilization is α , the clock frequency is F , and the off-chip memory bandwidth is B . The threshold of CTC ratio C_0 can be calculated as follows:

$$C_0 = \frac{\alpha NF}{B} \quad (1)$$

The theoretical CTC value of the convolutional layer in a dynamic neural network can be calculated by:

$$CTC = \frac{2K^2 W_o H_o \gamma_i C_i \gamma_o C_o}{W_i H_i \gamma_i C_i + W_o H_o \gamma_o C_o + K^2 \gamma_i C_i \gamma_o C_o} \quad (2)$$

where W and H denote the width and height, C denotes the number of channels, K denotes the kernel size and γ denotes the activation ratio of channels at the current layer. The subscripts i and o represent the input and output feature maps, respectively.

Since it is not known in advance which channels will be activated, all input channels will inevitably be loaded. With the reduction of computation, the CTC of the convolutional layer decreases further. Fig.3 statistically shows the theoretical CTC values of each layer in the static network and the dynamic network (the activation ratio of each layer of the dynamic network is averaged on the LLVIP dataset [20]). It can be seen that the system is memory-bounded.

In convolutional layers, the output pixel correlates with part of the input pixels. So we can get the output pixel directly without computing all the intermediate features. We propose the cross-layer scheduling for dynamic neural networks, so that the off-chip memory access of intermediate layers can be reduced and CTC can be improved.

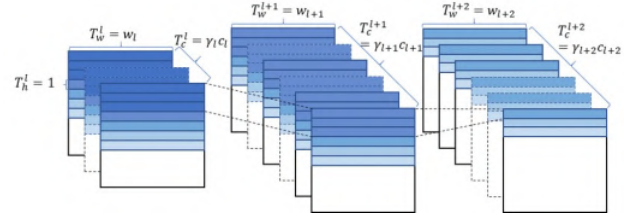


Fig. 4: Cross-layer scheduling of two convolutional layers.

Algorithm 1 Cross-Layer Scheduling for Dynamic Neural Networks

Input: Input Model m_0 , Number of layers L , number of DSPs provided N , DSP utilization α , Clock frequency F , Off-chip memory bandwidth B , activation ratio of each layer γ^l

Output: A collection of fused models M

$$C_0 = \frac{\alpha NF}{B}$$

$M = \{\}$

function FuseModel (m, M)

$CTC = \text{CalculateCTC}(m, \gamma^l)$

if $\forall set \text{ in } \{set_i\}, \text{length}(set) == 1$

or $\forall l \in set, s.t. CTC(l) \geq C_0$

append m **to** M

else

for $set \text{ in } \{set_i\}$ **and** $\text{length}(set) > 1$:

for $l \text{ in } set$ **and** $CTC(l) < C_0$:

$m_1 = \text{UpdateModel}(m, l, l+1)$

FuseModel(m_1, M)

$m_2 = \text{UpdateModel}(m, l, l-1)$

FuseModel(m_2, M)

end if

end function

FuseModel(m_0, M)

Fig.4 illustrates the cross-layer scheduling of two convolutional layers, and the kernel sizes are 1×1 and 3×3 . The dashed lines represent the pruned channels. Data has to be tiled due to limited on-chip memory, and T_h, T_w, T_c denote the tile shape of height, width, and channels, respectively. If the data is tiled on C dimension, then extra partial sums will be produced, resulting in $4 \times$ memory overhead (the features are 8-bit quantized and partial sums are 32-bit). In order to benefit the DMA transactions with continuous data, we cache the entire row in the on-chip SRAM. Therefore, it is more appropriate to tile on H dimension.

C. Design Space Exploration

As the conventional dynamic neural network processes different samples, its network structures are very different, so it is difficult to make a static analysis and determine the fusion strategy. To facilitate analysis, we set the model activation ratio to γ . The total computation of the static network is recorded as M_0 , and the computation of each layer is recorded as m_0^l . In the training phase, we count the actual computation of each sample after layer skipping and dynamic pruning, which is

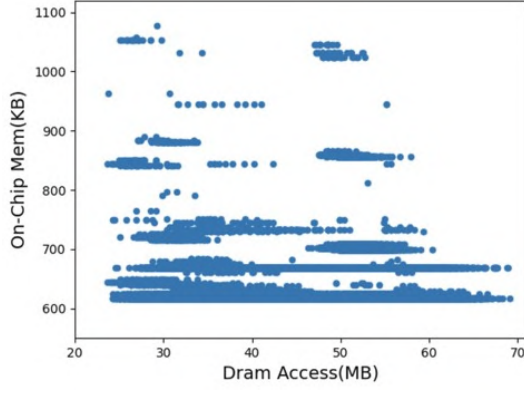


Fig. 5: Design space exploration of dynamic neural work.

recorded as M , and the actual computation of each layer is recorded as m^l . We divided the training into two steps. In the first step, we add a new item to the loss function:

$$L = L_0 + \theta \text{Abs}(\gamma - \frac{M}{M_0}) \quad (3)$$

where θ is a configurable parameter. We train the model until convergence and then obtain weights that make the overall activation ratio of the dynamic network close to γ . In the second step, we select a batch of samples for inference, count the actual activation ratio of each layer, and average them. Then we obtain the average activation ratio of each layer γ^l . The loss function is modified as follows:

$$L = L_0 + \theta \sum_l \text{Abs}(\gamma^l - \frac{m^l}{m_0^l}) \quad (4)$$

We train the model again. Finally, while ensuring accuracy, the activation ratio of each layer is close to the fixed value γ^l .

Algorithm 1 illustrates the design space exploration for cross-layer scheduling of dynamic networks. First, we calculate the threshold C_0 according to equation (1). Then, according to the model m and the activation ratio γ^l , the theoretical CTC value of each layer can be calculated. In any subset *set* which consists of at least two convolutional layers, an arbitrary layer with CTC less than C_0 is selected and fused with the previous or next layer to get the fused model m . Then the fused model is sent to the next iteration and the above operation is repeated. When either of the following two conditions is met, 1) all the convolutional layers in one subset are fused; 2) the theoretical CTCs of all layers are greater than the threshold C_0 , we terminate the iteration and obtain a fusion model m . Repeat the exploration and try different fusion branches. After traversing all the search spaces, we can get a collection M . In the collection, the size of on-chip memory and the DRAM access is different, depending on the structure of these models. A scatter plot can be drawn, as shown in Fig.5. Among them, the relatively good implementation in both indicators is selected as the final fusion model.

During the inference, the activation ratio of each layer may fluctuate, which affects the performance of the cross-

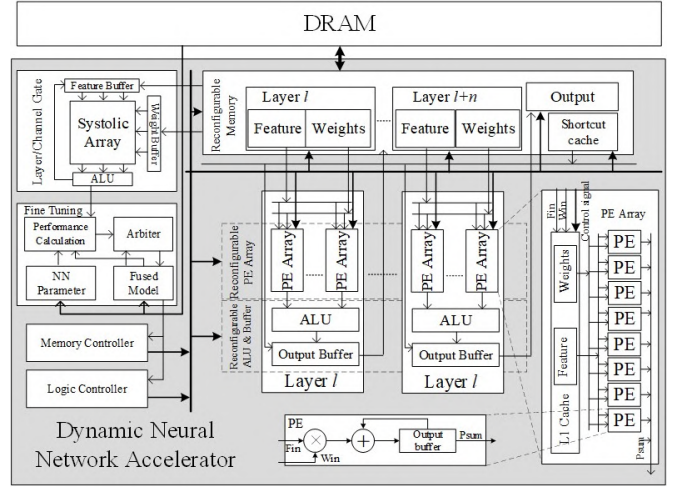


Fig. 6: Architecture of dynamic neural network accelerator.

layer scheduling. So we design a fine-tuning module in the accelerator. The fusion strategy is fine-tuned based on the actual activation ratio computed at runtime. The fine-tuning algorithm limits the search space and only fine-tunes 2 layers at most, so the computational overhead is minimal. The arbitrator can judge the fine-tuned model, and select the fusion model with the best performance to replace the previous model.

IV. HARDWARE ARCHITECTURE

A. Controller

Layer/Channel Gate is a systolic array of size 10×10 used to perform MLP layers. Feature maps flow vertically and weights are accessed independently by the PEs. The results are accumulated in the local buffers and finally accumulated in each column. After the accumulation is completed, results are sent to the ALU unit for activation, quantization, and other operations. The activation information of layers and channels is sent to the fine-tuning module.

Fine-tuning module consists of four parts: 1) Performance calculation module; 2) Arbitrator module; 3) Neural network parameters cache 4) Fused model cache. Before processing the layers, the parameters of each layer of the network and the fused model are loaded from DRAM to the cache. The performance calculation module obtains the activation ratio of each layer at runtime, loads the network parameters and the fusion model, and then calculates the indicators such as CTC and off-chip memory access of the fine-tuned model. Since the search space is further limited, the amount of computation here is small and does not affect the overall performance. The search results are fed into the arbitrator, which selects the optimal model and updates the fused model cache.

Memory controller and *logic controller* read the fusion model from the fused model cache. According to the specific parameters of the fusion layer, reconfigurable memory and computing resources are allocated to each layer.

TABLE I: COMPARISON WITH DIFFERENT IMPLEMENTATIONS.

	Computation(GOP)	DRAM access(MB)	BRAM(KB)	DSPs	Bandwidth(GB/s)	Clk(MHz)	Latency(ms)	FPS
baseline1	5.59	103	504	2800	12.8	160	10.6	94.7
baseline2 [15]	5.59	35	576	2800	12.8	160	6.8	146.1
ours	3.28	24	612	2800	12.8	160	3.9	256.8

TABLE II: TRAINING RESULTS ON LLVIP DATASET.

Model	mAP50	mAP75	mAP
static two-stream MobileNetV2-YoloV5	93.4	68.5	59.2
dynamic two-stream MobileNetV2-YoloV5	93.2	68.1	58.9

B. Processing Module

To adapt to different cross-layer scheduling strategies, we introduce a reconfigurable processing module composed of fine-grained PE arrays, ALUs, and buffers. Each PE array consists of 8 PEs. The PEs in the PE array are cascaded to each other. Data flow can be transmitted and broadcast among the PE arrays and PEs for data reuse.

All the configurable units are divided into multilayers. The division is according to the proportion of the computational cost of different layers so that the throughput of different layers is balanced. With the appropriate division, the intermediate data can be consumed by the next layer in time. When the processing module starts working, the PE arrays read the features and weights from the SRAM and share them with different PEs.

PEs produce the partial sums and pass them to the Arithmetic Logic Units (ALUs). The partial sums are accumulated in the ALUs. After the accumulation of partial sums is completed, the ALUs will re-quantize the output data from 32-bit to 8-bit. If the shortcut structure is activated, ALUs will also read the features from the shortcut cache and accumulate them with partial sums. Then the output will be cached in the output buffer and passed to the next layer.

V. EVALUATION

A. Implementation Details

The static and dynamic multispectral detection models are evaluated on the LLVIP dataset [20]. We use the Stochastic Gradient Descent (SGD) optimizer with an initial learning rate of $1e-2$, a momentum of 0.937, and a weight decay of 0.0005. To avoid optimization instabilities, we use the first three epochs for warmup. The warmup initial momentum is set to 0.8 and the warmup initial bias learning rate is set to 0.1. For data augmentation, we use the mosaic method which mixes four training images into one image. The models are developed on an Ubuntu 18.04 platform with PyTorch 1.12.0 and two NVIDIA 3090 GPUs. The networks are trained for 20 epochs and the batch size is 32.

The dynamic neural network accelerator is evaluated on the dynamic two-stream MobileNetV2 backbone. All the input images will be resized to 256×256 during preprocessing. We compare the performance with other FPGA implementations

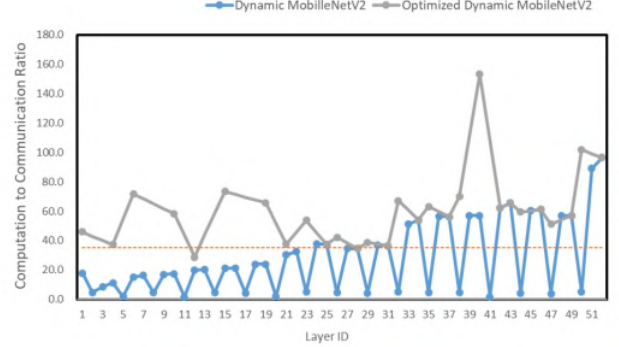


Fig. 7: The CTCs of optimized dynamic neural network.

without optimization and with cross-layer scheduling on the static network [15]. The accelerators are implemented on a Xilinx VC707 board, which contains 2800 DSP slices, 4635KB SRAM and 12.8GB/s 1GB DDR3. All the features and weights are quantized to Int8.

B. Results

Fig.7 shows the CTCs of the optimized dynamic neural network. After applying cross-layer scheduling, the CTCs of the optimized dynamic neural network are greatly improved, exceeding the threshold value $C_0 = 35$ of VC707. Therefore, the performance of the optimized model will not be limited by DRAM bandwidth, and DSP resources can be fully utilized.

Table I shows the comparison with other baseline implementations. Benefiting from layers skipping and channel pruning, our dynamic neural network accelerator reduces the computational overhead by 41% and DRAM access by 77% compared to the unoptimized baseline1. Compared with baseline2, which only supports cross-layer scheduling for static networks, our implementation reduces the computational overhead by 41% and DRAM access by 31%. Therefore, the inference performance of our implementation is greatly improved by $1.8 \times - 2.7 \times$ compared to the baseline. Due to the addition of the Layer/Channel Gate module, we increase the on-chip memory by an additional 6%-21% compared to the baseline. Given the huge performance improvement, a small increase in on-chip memory is acceptable.

Table 2 shows the training result of dynamic and static networks on the multispectral detection dataset LLVIP. It can be seen that the accuracy of the dynamic network in mAP50, mAP75, and mAP does not decrease by more than 1% compared with the static network.

VI. CONCLUSION

In this paper, we propose an efficient dynamic neural network for multispectral detection. Compared with the unoptimized static model, our proposed model reduces computation by 41% and off-chip memory access by 77% on average with less than 1% decrease in mAP. We design a cross-layer scheduling for dynamic neural networks to improve the computation to communication (CTC) ratio. Furthermore, we designed the scheduling arbitration module, which can fine-tune fusion strategy according to the activation ratio of each layer at runtime to obtain the optimal scheduling scheme. We propose a flexible hardware architecture design for dynamic neural networks. Experiments demonstrate that our dynamic neural network accelerator achieves a maximum $2.7\times$ inference performance improvement.

REFERENCES

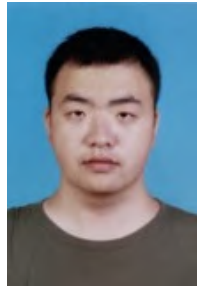
- [1] J. W. Davis and V. Sharma, "Background-subtraction using contour-based fusion of thermal and visible imagery," *Computer vision and image understanding*, vol. 106, no. 2-3, pp. 162–182, 2007.
- [2] W. Ouyang and X. Wang, "Joint deep learning for pedestrian detection," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 2056–2063.
- [3] X. Li, L. Li, F. Flohr, J. Wang, H. Xiong, M. Bernhard, S. Pan, D. M. Gavrilu, and K. Li, "A unified framework for concurrent pedestrian and cyclist detection," *IEEE transactions on intelligent transportation systems*, vol. 18, no. 2, pp. 269–281, 2016.
- [4] M. Bilal, A. Khan, M. U. K. Khan, and C.-M. Kyung, "A low-complexity pedestrian detection framework for smart video surveillance systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 10, pp. 2260–2273, 2016.
- [5] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, "Squeezednet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 129–137.
- [6] Z. Shao, G. Cheng, J. Ma, Z. Wang, J. Wang, and D. Li, "Real-time and accurate uav pedestrian detection for social distancing monitoring in covid-19 pandemic," *IEEE transactions on multimedia*, vol. 24, pp. 2069–2083, 2021.
- [7] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.
- [8] T. Geng, C. Wu, C. Tan, C. Xie, A. Guo, P. Haghi, S. Y. He, J. Li, M. Herbordt, and A. Li, "A survey: Handling irregularities in neural network acceleration with fpgas," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2021, pp. 1–8.
- [9] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 409–424.
- [10] A. Veit and S. Belongie, "Convolutional networks with adaptive inference graphs," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 3–18.
- [11] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-z. Xu, "Dynamic channel pruning: Feature boosting and suppression," *arXiv preprint arXiv:1810.05331*, 2018.
- [12] C. Herrmann, R. S. Bowen, and R. Zabih, "Channel selection using gumbel softmax," in *European Conference on Computer Vision*. Springer, 2020, pp. 241–257.
- [13] B. E. Bejnordi, T. Blankevoort, and M. Welling, "Batch-shaping for learning conditional channel gated networks," *arXiv preprint arXiv:1907.06627*, 2019.
- [14] J. Chen, Z. Zhu, C. Li, and Y. Zhao, "Self-adaptive network pruning," in *International Conference on Neural Information Processing*. Springer, 2019, pp. 175–186.
- [15] X. Wang, T. Tian, L. Zhao, W. Wu, and X. Jin, "Exploration of balanced design in resource-constrained edge device for efficient cnns," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2022.
- [16] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, 2015, pp. 161–170.
- [17] G. Jocher, "Yolov5 release v5.0," 2022. [Online]. Available: <https://github.com/ultralytics/yolov5/releases/tag/v5.0>
- [18] K. Zhou, L. Chen, and X. Cao, "Improving multispectral pedestrian detection by addressing modality imbalance problems," in *European Conference on Computer Vision*. Springer, 2020, pp. 787–803.
- [19] Z. Chen, Y. Li, S. Bengio, and S. Si, "You look twice: Gatnet for dynamic filter selection in cnns," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9172–9180.
- [20] X. Jia, C. Zhu, M. Li, W. Tang, and W. Zhou, "Llvp: A visible-infrared paired dataset for low-light vision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 3496–3504.



Xiaotian Wang received his B.S. degree in 2017 from University of Science and Technology of China, and he is currently a Ph.D. student in Department of Physics in University of Science and Technology of China, Anhui, China, under the supervision of Prof. Xi Jin. His current research work is mainly on FPGA-based Hardware Accelerator design.



Letian Zhao received his B.S. degree in 2017 from University of Science and Technology of China, and he is currently a Ph.D. student in Department of Physics in University of Science and Technology of China, Anhui, China, under the supervision of Prof. Xi Jin. His current research work is mainly on FPGA-based Hardware Accelerator design.



Wei Wu received his B.S. degree in 2016 from University of Science and Technology of China, and he is currently a Ph.D. student in Department of Physics in University of Science and Technology of China, Anhui, China, under the supervision of Prof. Xi Jin. His current research work is mainly on SSD-based Accelerator design.



Xi Jin received the B.S. degree from University of Science and Technology of China, and he is currently an associate professor in Department of Physics in University of Science and Technology of China, Anhui, China. His research interests include SOC design technology, VLSI design, computer-aided design methodologies for SoC system integration

and FPGA-based Hardware structure design.