Enabling Integrity Measurement for Secure Applications in the Enarx Framework

Accelerating and Pruning CNNs for Semantic Segmentation on FPGA

Pierpaolo Morì[*1,2], Manoj-Rohit Vemparala[2], Nael Fasfous[3], Saptarshi Mitra[3], Sreetama Sarkar[3], Alexander Frickenstein[2], Lukas Frickenstein[2], Domenik Helms[4], Naveen Shankar Nagaraja[2], Walter Stechele[3] and Claudio Passerone[1]

[1]*Department of Electronics and Telecommunications, Politecnico Di Torino, Turin, Italy*
[2]*Autonomous Driving, BMW Group, Munich, Germany*
[3]*Department of Electrical and Computer Engineering, Technical University of Munich, Munich, Germany*
[4]*Deutsches Zentrum für Luft und Raumfahrt, Munich, Germany*

{<firstname>.<lastname>} [1]@polito.it,[2]@bmw.de,[3]@tum.de,[4]@dlr.de

ABSTRACT

Semantic segmentation is one of the popular tasks in computer vision, providing pixel-wise annotations for scene understanding. However, segmentation-based convolutional neural networks require tremendous computational power. In this work, a fully-pipelined hardware accelerator with support for dilated convolution is introduced, which cuts down the redundant zero multiplications. Furthermore, we propose a genetic algorithm based automated channel pruning technique to jointly optimize computational complexity and model accuracy. Finally, hardware heuristics and an accurate model of the custom accelerator design enable a hardware-aware pruning framework. We achieve 2.44× lower latency with minimal degradation in semantic prediction quality (−1.98 pp lower mean intersection over union) compared to the baseline DeepLabV3+ model, evaluated on an Arria-10 FPGA. The binary files of the FPGA design, baseline and pruned models can be found in github.com/pierpaolomori/SemanticSegmentationFPGA

1 INTRODUCTION

Convolutional neural networks (CNN) are widely used for solving problems like image classification [1], object detection [2], and semantic segmentation [3]. The deployment of these networks on resource-limited hardware (HW), such as in-vehicle systems, remains challenging due to high memory requirements and energy consumption. GPUs are most favorable during training, but often in case of inference, FPGAs [4] and ASICs [5] outperform GPUs in terms of power efficiency. In this work, we focus on the deployment of a state-of-the-art semantic segmentation-based CNN, namely the DeepLabV3+ [3] model, on reconfigurable hardware. The primary motivation is to design an efficient accelerator using high-level synthesis (HLS) to meet the high compute and memory requirements of a substantially complex model like DeepLabV3+. To further facilitate the efficient execution on HW, we implement a HW-aware genetic algorithm (GA) to search for Pareto optimal pruned CNNs. We address the challenge of designing *HW-friendly CNN architectures* by integrating a HW model of the designed semantic segmentation accelerator into the pruning search. We further refine the pruning configurations by deriving HW heuristics to guide the search algorithm while compressing the network.
We summarize our contributions as follows:

(1) **Accelerator Design for Segmentation.** We implement an *end-to-end* semantic segmentation application using DeepLabV3+ [3] on an FPGA. We enable support for 2D dynamic tiling, 3D unrolling, dilated convolutions, and bilinear upsampling to an existing accelerator design. Our accelerator achieves 90% DSP utilization, 183.3 GOPS throughput, and DRAM accesses reduction by a factor of 2.33×.
(2) **Genetic Algorithm-based Channel Pruning.** We formulate a non-dominated sorting genetic algorithm (NSGA-II) to determine Pareto optimal pruning configurations. We obtain 2.75× reduction in the number of operations with minimal degradation in prediction quality.
(3) **Hardware-Aware Pruning.** We formulate an analytical HW model of our accelerator to steer a *latency-driven* GA search and outperform pruning configurations based on proxy metrics, such as the number of operations. The latency-driven GA search provides a performance improvement of 2.44×, with minimal degradation in the prediction quality.

The remainder of the paper is structured as follows: Sec. 2 discusses related work on FPGA acceleration and structured pruning. Sec. 3 introduces our approach to an end-to-end deployment pipeline of the DeepLabV3+ model on an Arria-10 GX FPGA, which is then extensively evaluated in Sec. 4. Sec. 5 concludes the paper.

2 RELATED WORK

2.1 CNN Acceleration with FPGA

Different approaches try to overcome the limitations of constrained FPGA platforms, improving inference efficiency [6–9]. Zhang et al. [6] formulate a uniform matrix multiplication based representation for convolutional and fully-connected layers. They perform batch processing to tackle the memory bandwidth problem in fully-connected layers. Wang et al. [7] proposed PipeCNN, that adopts a pipelined data structure using multiple functional units. The input features and weights are fetched from DRAM in a double buffer by sliding a 3D window. Although PipeCNN achieves high throughput, there is no further optimization to reduce DRAM access and minimize memory stalls. Ma et al. [8] exploit loop optimization techniques such as tiling and unrolling. The schedule for optimal data reuse patterns depends on the CNN's layer dimensions and tiling factors. The unrolling (or parallelization) factors directly impact the compute utilization, improving the latency of the convolution execution. Bai et al. [9] designed an FPGA accelerator for semantic segmentation to speed up inference time of SegNet-basic model [10].

[*] Pierpaolo Morì has contributed to this work during his master thesis at BMW AG..

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530424

They implemented transpose convolution efficiently to avoid redundant zero multiplications. However, loop tiling optimization is limited to the input channel dimension, and the processing element is suitable only for 3×3 and 2×2 kernels, restricting the flexibility of the accelerator. We extended the work of PipeCNN [7] by adding support to bilinear upsampling and dilated convolutional layers to accelerate segmentation models. We improve the performance using 3D loop unrolling and dynamic tiling techniques to increase resource utilization and reduce DRAM accesses.

2.2 Structured CNN Pruning

Han et al. [11] determined the saliency of weights based on their magnitude. Pruning individual weights, referred to as irregular pruning, leads to inefficient memory accesses, requiring specialized implementations. Regularity in pruning becomes an important criterion when considering accelerator-aware optimization. He et al. [12] proposed structured channel pruning, where the saliency of individual channels is determined through Lasso regression. The pruning rate for each layer is based on handcrafted heuristics, which target lower *proxy* metrics such as `OPs` and `Params`. The reinforcement learning (RL) agent in AMC [13] automates the search for optimal pruning rates in each layer. The RL-agent produces pruning configurations based on real latency measurements on smartphone devices, i.e. necessitating a HW-in-the-loop (HIL) setup. Although the RL-agent in AMC reduces the dependency on human experts, it still requires a handcrafted reward function due to its single-objective search approach. In this work, we propose a GA-based pruning method to improve on existing techniques and determine Pareto optimal compressed solutions. We remove the necessity of a HIL-based approach, which is not always feasible when the HW is not synthesized or readily available. To maintain HW-awareness, we develop an *analytical* HW model to replace the HIL, providing a HW *model-in-the-loop* approach. Additionally, we use the multi-objective search algorithm, NSGA-II, to relieve the requirement of handcrafting a reward function, which may not always fairly balance all objectives (accuracy, latency, parameter count, etc.).

2.3 Evolutionary Channel Pruning

Yang et al. [14] proposed a GA search for CNN channel pruning. Each individual in the population represents a slim variant of the pretrained CNN model, pruned with different combinations. Individuals are selected for the next population according to a multi-objective trade-off among accuracy degradation, number of operations, and pruned values. At every search step, the individual is fine-tuned to compensate for the loss of accuracy. Hardware metrics are not considered in the GA search, and the investigation is limited to small-scale datasets such as MNIST. Hu et al. [15] presented a pruning procedure based on a layer-wise GA, which identifies redundant filters. The CNN model is pruned layer by layer sequentially using fixed compression ratios based on sensitivity heuristics. At each iteration, fine-tuning based on knowledge distillation is performed to restore the model accuracy. However, their approach is not scalable to huge models, such as DeepLabV3+, due to the need for many GPU hours during the search phase. Instead, our GA searches for effective compression ratios and reduces the GPU hours by relying on a magnitude-based heuristic. Wang et al. [16] introduced APQ, a methodology to jointly perform neural architecture search, pruning, and quantization. A quantization-aware accuracy predictor and a lookup table containing hardware metrics for each layer is used to estimate accuracy and latency respectively. Although the approach reduces the number of operations, the HW optimization techniques are not considered to reduce latency. For example, specialized layers required for semantic segmentation, such as dilated convolutions, require novel dataflows to optimize memory demand and latency. In our work, we incorporate specialized HW blocks for semantic segmentation *and* perform HW-aware pruning to reduce latency with minimal degradation of mIOU.

3 ACCELERATING SEMANTIC SEGMENTATION

We propose a specialized FPGA accelerator with an end-to-end compression pipeline using structured channel pruning to reduce the latency of semantic segmentation, as shown in Fig. 1. The word length of weights and activations is set to 16-bit to maximize the DSP's capabilities and limit the numerical error introduced by low-bit quantization. In Sec. 3.1, we introduce the architecture description of the accelerator design. In Sec. 3.2, a theoretical model of our accelerator is detailed to predict latency. Finally, in Sec. 3.3, we highlight an automated channel pruning framework using a genetic algorithm, which determines suitable compression ratios for a semantic segmentation based CNN.

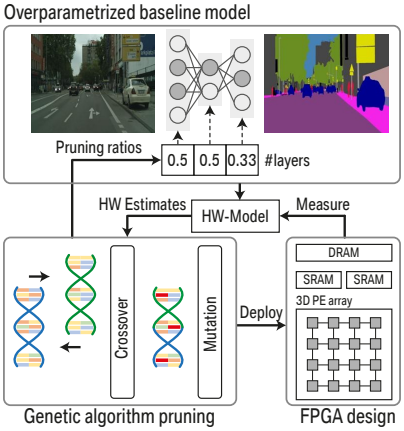

Figure 1: End-to-end CNN deployment pipeline on an FPGA platform using genetic algorithm-based channel pruning.

3.1 FPGA Accelerator Design

The high-level design description of the proposed FPGA accelerator is presented in Fig. 2.

Data Buffers and Dynamic Scheduling: The accelerator design employs input, weight, and output buffer subsystems directly connected to the external DRAM. The complexity of the accelerator depends on the unrolling factors P_{of}, P_{if}, P_{kx} for the output filters, input channels, and kernel dimension, respectively. At every clock cycle, P_{if} input features and $P_{kx} \times P_{if} \times P_{of}$ weights are fetched from the external DRAM and stored in the on-chip input and weight buffers, respectively. P_{of} output features are written back. Performing three global memory accesses at each MAC operation would cause a huge degradation in accelerator performance due to limited bandwidth. Loading all the needed data into on-chip memory is

generally not possible due to the limited storage size. Since the

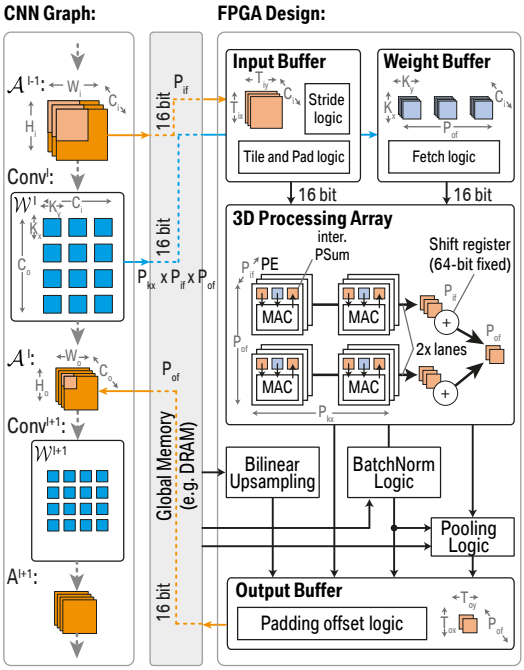

Figure 2: Overview of semantic segmentation based FPGA design: Two dimensional tiling factors T_{ix}, T_{iy} are determined dynamically for a CNN layer to improve the reuse of input features. PE array performs unrolling in three dimensions with $P_{of} \times P_{if} \times P_{kx}$ multipliers.

neighboring outputs of the convolution operations share input features, redundant data accesses can be avoided by performing the convolution in tiles (loop tiling). According to T_{ox} and T_{oy}, dynamically defined in real time for each layer, the proposed accelerator tiles $T_{ix} \times T_{iy} \times C_i$ input feature maps, $k_x \times k_y \times C_i \times P_{of}$ weights, generating $T_{ox} \times T_{oy} \times P_{of}$ output features. Here, $T_{ix}, T_{iy}, T_{ox}, T_{oy}$ represent the tiling factors for input and output feature maps, C_i represents the number of input channels, k_x and k_y represent the kernel matrix dimensions.

Processing Element Array: The parallel convolutions are performed in the PE array consisting of interconnected DSPs. The convolution operation can be unrolled in three dimensions, as demonstrated in Fig. 2. Partial sums are stored in a shift register with configurable depth d, representing the latency of the MAC operation (in cycles). Each partial sum S_n generated at cycle n is accumulated with the partial sum S_{n-d}, d cycles before. For the 16-bit CNN inference on the Intel Arria 10 board, 2 MACs can be mapped to a single DSP block, therefore $\frac{1}{2} \cdot P_{if} \cdot P_{of} \cdot P_{kx}$ DSP blocks are needed. For lower data bit-width, the amount of DSP utilization does not change. The PE array generates P_{of} independent partial sums at each clock cycle of the accelerator.

Additional HW and Semantic Segmentation: The pooling logic (Max and Average) consists of comparator, adder and divider units. It buffers the first rows or partial pooling results until the whole needed input window is received and can directly fetch data from DRAM. The batch normalization kernel unit receives 16-bit fixed-point inputs from the convolution kernel. These are converted to floating-point, normalized using the layer parameters (mean, variance, gamma, and beta, directly loaded from DRAM), and converted back to fixed-point. In order to improve flexibility, we implemented batch normalization as a dedicated unit, instead of fusing it with convolution. For semantic segmentation, dilated convolution is commonly used to encode multi-scale contextual information. The weight kernels are inflated with dummy zeros to get a larger projection area for the dilated convolutions on the input feature map. For higher dilation rates, the weight buffer size explodes, demanding large on-chip memory to implement that particular layer. Therefore, instead of extending the weight kernel dimensions, our dataflow selects specific activation values based on the dilation rate. In order to resize the downsampled features to the original image, we designed an upsampling implementation using bilinear interpolation. Based on the description of the CNN model and the requirements for each layer, the HW connections in the FPGA are reconfigured. This allows to hugely increase the flexibility of our accelerator and speed up the inference of several CNN models (*e.g.*, ResNet18, ResNet34, DeepLabV3+) for different applications (*e.g.*, image classification, semantic segmentation).

3.2 Theoretical Hardware Model

We model the FPGA-based accelerator design to determine real-time metrics such as latency and DRAM accesses, and facilitate the CNN pruning process without requiring the synthesized FPGA design in the genetic algorithm loop.

Optimal Tiling Factors: For estimating the latency of the convolution with stride s, it is important to determine if the layer execution is compute or memory bounded. Using Eq. 1, we maximize T_{ox} and T_{oy} dynamically such that the tiled input features fit on the allotted input buffer $\texttt{Buf}_{in}$, thereby reducing DRAM accesses.

$$\underset{T_{ox},T_{oy}}{\arg\max} \left(\frac{T_{ox} \cdot T_{oy} \cdot k_x \cdot k_y \cdot P_{of}}{((T_{ox}-1)s + k_x) \cdot ((T_{oy}-1)s + k_y)} \right) \\ \ni ((T_{ox}-1)s + K_x) \times ((T_{oy}-1)s + K_y) \times C_i \leq \texttt{Buf}_{in} \tag{1}$$

Compute Bounded Latency: The required number of cycles $Lat_{compute}$ for the unrolled computation per layer is given by Eq. 2. Here, W_o and H_o represent the spatial dimensions of the output feature maps.

$$Lat_{compute} = \left\lceil \frac{C_i}{P_{if}} \right\rceil \cdot \left\lceil \frac{k_x}{P_{kx}} \right\rceil \cdot \left\lceil \frac{C_o}{P_{of}} \right\rceil \cdot k_y \cdot W_o \cdot H_o \tag{2}$$

Memory Bounded Latency: We estimate the number of DRAM access for input, weights and output to calculate the memory bounded latency Lat_{mem}. Our accelerator reduces the DRAM accesses by tiling $T_{ix} \cdot T_{iy} \cdot C_i$ input features in the on-chip buffer. The total number of input DRAM accesses is formulated as product of number of tiles and tile size as shown in Eq. 3.

$$DRAM_{in} = \underbrace{\left\lceil \frac{W_o}{T_{ox}} \right\rceil \cdot \left\lceil \frac{H_o}{T_{oy}} \right\rceil \cdot \left\lceil \frac{C_o}{P_{of}} \right\rceil}_{\text{number of tiles}} \cdot \underbrace{T_{ix} \cdot T_{iy} \cdot C_i}_{\text{input tile size}} \tag{3}$$

where, $T_{ix} = (T_{ox}-1)s + k_x$ *and* $T_{iy} = (T_{oy}-1)s + k_y$

We maximize the reuse of weights and output features on-chip to prevent *re-fetching/writing* these data types onto the external DRAM. The DRAM accesses due to weights is given by $DRAM_w = k_x \cdot k_y \cdot C_i \cdot C_o$ and outputs is given by $DRAM_{out} = W_o \cdot H_o \cdot C_o$. The resulting memory bounded latency due to DRAM access is expressed in Eq. 4.

$$\#DRAMAccess = DRAM_{in} + DRAM_w + DRAM_{out}$$
$$Lat_{mem} = \frac{\#DRAMAccess}{\texttt{MemoryBandwidth}} \tag{4}$$

We set $T_{ox} = 1$, $T_{oy} = 1$ for convolutional layers with dilation rate > 1, to avoid prohibitive input buffer requirements. Due to limited bandwidth (9.5 GB/s) on the Arria-10 FPGA, the memory bounded latency in dilated convolutional layers dominates the compute latency. Therefore, we avoid a huge memory demand for dilated convolutional layers.

Overall Latency: We calculate the total latency Lat_{tot} of convolutional layers by estimating the maximum number of cycles required for compute and memory transfer as shown in Eq. 5.

$$Lat_{tot} = \texttt{max}(Lat_{compute}, Lat_{mem}) \tag{5}$$

Estimating Latency for the Upsampling Layer: We estimate the latency of bilinear upsampling layers by performing linear interpolation with different number of output channels N_{of}, as shown in Fig. 3. Based on predictions of the linear interpolation (shown in blue), we estimate the latency of the two bilinear upsampling layers with various filter choices. Linear interpolation-based latency estimation is also leveraged for the average pooling layer in the ASPP module of the DeepLabV3+.

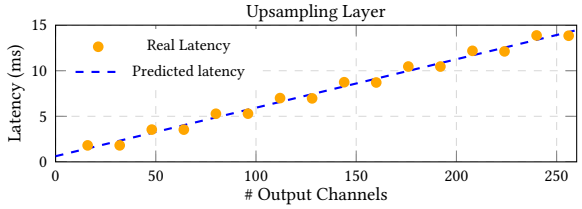

Figure 3: Linear regression-based latency estimation for the upsampling layer.

3.3 Genetic Algorithm-based Channel Pruning

We formulate the channel pruning process as a search problem, where redundant filters are pruned based on layer-wise compression ratios and a magnitude-based heuristic. The pruning rates result in an integer number of remaining channels for each layer. Pruning certain filters leads to large degradation in accuracy (mIOU), highlighting different sensitivities for various pruning choices. Moreover, pruning makes large portions of the total computations possible in a single tile, leading to schedule-dependent improvements in latency. These aspects make the search space *discrete* and *non-differentiable*. We leverage genetic algorithms (GAs) to tackle this search problem, as they are resilient to noisy search spaces, quick to prototype, and do not need smooth, continuous search spaces to perform well (gradient-free). We also relieve the burden of handcrafting a cost function to combine the multiple criteria (mIOU, latency, OPs) by using NSGA-II, which follows a multi-objective Pareto-optimal selection approach. This ultimately facilitates better design space exploration and maintains diverse non-dominated trade-off solutions $\mathcal{P}_{opt}$, which may fit different deployment scenarios.

Explicit, bijective encoding is used to create the genomes of potential solutions. A single genome represents a potential CNN pruning strategy and has as many genetic loci as layers in the CNN. Each genetic locus encapsulates a pruning rate at the corresponding layer. Sparsity-to-layer encoding can be captured intuitively in sequential genomes, providing a one-to-one application of GA operators onto the CNN layer sequence. Neighboring CNN layers have higher feature correlation than distant layers, therefore, pruned layer relationships encoded in neighboring genetic loci can survive in a population and be reused through single-point crossover to create more efficient offspring. The fitter the parents become throughout the generations, the better genetic localities they will have, potentially leading to better offspring. Mutation further allows offspring to escape local minima of their parents.

The GA search initially starts with a population size $|\mathcal{P}|$. Each individual in the population $\mathcal{P}$ is a CNN, with randomly selected pruning rates r of its layers. These networks are evaluated to obtain their mIOU and HW estimates. The mIOU values during the search are obtained by evaluating the pruned model on random samples of the train set, to maintain an unbiased population w.r.t. the validation set. The HW-related optimization criterion can either be the number of operations in the CNN, or the latency values which are calculated using the HW model discussed in Sec. 3.2. The individuals in the population then undergo crossover (with probability p_c) and mutation (with probability p_m) to produce offspring. We evaluate the mIOU and HW estimate values of each offspring. Based on their Pareto-optimality, selected parents and offspring survive into the next generation. This process is continued for g generations. The GA based pruning procedure is detailed in Alg. 1.

Algorithm 1: Channel Pruning using NSGA-II.

Initialize : Genome encoding, Population size $|\mathcal{P}|$, crossover probability p_c, mutation probability p_m, random pruning rates r, and maximum generation number g.
Input : Random initial population $\mathcal{P}$, pre-trained CNN.
Output : Pareto-optimal pruned solutions $\mathcal{P}_{opt}$, balancing accuracy and hardware estimates.
for *gen in range(1, g)* **do**
  Evaluation : Apply magnitude-based pruning. Evaluate the fitness of each individual in $\mathcal{P}$ according to mIOU and HW latency.
  Selection : Non-dominated sorting and crowding distance selection.
  Crossover : Select two individuals and perform single-point crossover with probability p_c.
  Mutation : Perform replace mutation on each offspring with probability p_m.
  Update population $\mathcal{P}$.
end

4 EXPERIMENTS

To evaluate the FPGA design and the mIOU, we use DeepLabV3+ [3] with ResNet18 [1] backbone trained on the CityScapes dataset [17] consisting of 2974 train and 500 validation images with a resolution of 1024×2048 pixels. We downsample the CNN's input resolution to 960×960 pixels. For the DeepLab-based CNN architecture, the

bottleneck layers consist of two residual blocks with a dilation rate of 2 and an Atrous Spatial Pyramid Pooling (ASPP) block with dilation rates {1, 8, 12, 18}. Total 19 classes were used for annotating segmentation output and the resulting baseline mIOU is 67.27%. If not otherwise mentioned, all hyper-parameters specifying the task-related training were adopted from the CNN's base model. The FPGA platform used is the Arria 10 GX 1150 with a 2GB DDR4 DRAM, where Intel OpenCL SDK 19.4 is used to synthesize and generate the bitstream. The host program is compiled and executed on Intel Xeon Gold 5222 processor which is responsible for preparing images and weights and loading them onto the FPGA's DRAM to perform CNN inference.

4.1 CNN Accelerator Design

Reducing External Memory Access: Reducing off-chip data movement and increasing utilization of the PEs results in improved energy efficiency and throughput. To reduce the number of DRAM accesses, we improve the reuse of input feature maps by dynamically selecting tiling factors for each layer or increasing the on-chip buffer size as detailed in Sec. 3.2. To understand the importance of different tiling schemes, we synthesize an accelerator with 128KB of BRAM to buffer input features and 1024 DSP blocks to perform convolution operations in the PE array.

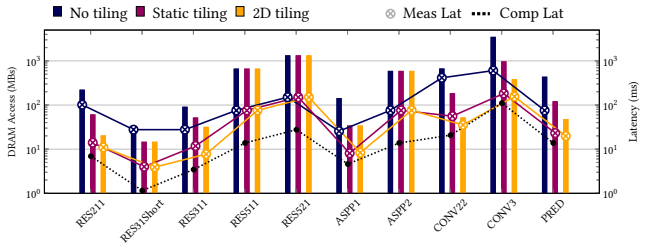

Figure 4: Evaluating different tiling schemes for convolutional layers with different spatial input dimensions.

We report DRAM accesses (bar graph) and latency (line graph) in Fig. 4 for different types of convolutional layers in the DeepLabV3+ model. We also highlight the effectiveness of the dynamic tiling scheme leveraged by our FPGA accelerator. We observe that the 2D dynamic tiling produces lower DRAM accesses for all the layers. Using the 2D tiling scheme, we reduce the overall DRAM accesses in the DeepLabv3+ model by a factor of 1.23× and 2.33× compared to the configuration with static and no tiling schemes respectively. We simulate the theoretical compute latency considering a target clock frequency of 200 MHz for each layer based on Eq. 2. We observe that the 2D tiling scheme produces similar measurement as compute bounded latency for non-dilated convolutional layers. We also observe that the dilated convolutional layers (RES511, RES521, ASPP2) do not obtain benefits in DRAM accesses using various tiling schemes as we set $T_{ox} = 1$ and $T_{oy} = 1$.

Unrolling the Convolution: Table 1 shows the resource utilization overhead of our accelerator that supports semantic segmentation w.r.t. our standard implementation for image classification for different unrolling configurations (P_{of}, P_{if}, and P_{kx}) on the Arria 10 FPGA. We observe greater HW utilization (+6% logic, +14% BRAM and 2% DSP) to accelerate DeepLabV3+ due to the additional layers supported (described in Sec. 3.1). We also report the inference latency for ResNet18 and DeepLabV3+ for different unrolling configurations. Using the unrolling factors $P_{if} = 16$, $P_{of} = 32$, $P_{kx} = 4$, we obtain a latency of 16.4ms and 1.6s for ResNet18 and DeepLabV3+ respectively.

Table 1: Resource utilization and latency of segmentation and classification accelerators for different unrolling configurations.

Unroll config P_{if}, P_{of}, P_{kx}	Segmentation (DeepLabV3+) Logic Util	DSP blocks	BRAM M20K (%)	Lat (s)	Classification (ResNet18) Logic Util	DSP blocks	BRAM M20K (%)	Lat (ms)
16, 16, 1	41%	394 (26%)	35%	3.9	33%	321 (21%)	23%	51.8
16, 32, 1	58%	690 (46%)	64%	2.5	46%	569 (37%)	32%	28.3
16, 16, 4	51%	778 (51%)	49%	2.2	43%	704 (46%)	41%	25.1
16, 32, 4	72%	1362 (90%)	75%	1.6	66%	1336 (88%)	61%	16.4

4.2 Channel Pruning using Genetic Algorithm

We conduct a hyper-parameter study for NSGA-II to understand the characteristics of the pruning search space, and the relationship between layer-wise sparsity, number of operations and mIOU. For all experiments, we set mutation (p_m) and crossover (p_c) probabilities to 0.4 and 1, respectively. In Fig. 5, we study the pruning performance by varying the hyper-parameters of the population size $|\mathcal{P}|$ and number of generations g. Increasing $|\mathcal{P}|$ and g demands more GPU hours to obtain Pareto optimal pruning configurations (indicated by the red line). For example, the GA search with ($|\mathcal{P}|$, g) = (10, 10) requires a search time of 0.7 hours using a single NVIDIA Volta GPU. For comparison, ($|\mathcal{P}|$, g) = (50, 50) requires a search time of 16 hours. Analysing the resulting Pareto-fronts of each experiment in Fig. 5, we notice that the (10, 10) configuration produces a limited Pareto-front, with larger gaps between the solutions. Increasing the size of the experiment to (25, 25) produces a much wider and dense Pareto-front. Beyond that, experiments of (25, 50) and (50, 50) result in less impressive improvements, making the (25, 25) configuration a good trade-off in terms of GPU hours and resulting Pareto-front. For ($|\mathcal{P}|$, g) = (25, 25), we fine-tune uniformly sampled Pareto optimal solutions (indicated by the green dots), and obtain a 2.75× reduction in number of operations with minimal degradation in the mIOU (-1.5pp).

Further, we compare the pruning configurations of the GA search with a state-of-the-art RL-based pruning [13] in Fig. 6. For the sake of comparison, we modify the accuracy guaranteed reward function in [13] to obtain different convergence regions for the DDPG-based RL agent. We observe that the NSGA-II based GA search obtains Pareto dominant solutions, whereas RL-based search focuses only on a specific region in the search space. For most of the compression ratios in Fig. 6, we observe that the GA-based pruned solutions dominate RL search in terms of mIOU and operations.

4.3 Hardware aware pruning

Proxy HW metrics, such as operation count (OPs), does not always guarantee tangible improvements on measured hardware estimates. Thus, we incorporate the latency estimate produced by the theoretical model explained in Sec. 3.2 into the GA based pruning search. We also improve hardware benefits by ensuring the same channel pruning rate to the layers with common input features (e.g. shortcut layers in the residual blocks and parallel convolutions in ASPP blocks). This facilitates maximum utilization of DRAM bandwidth and compute throughput of the FPGA accelerator towards producing optimal semantic prediction quality.

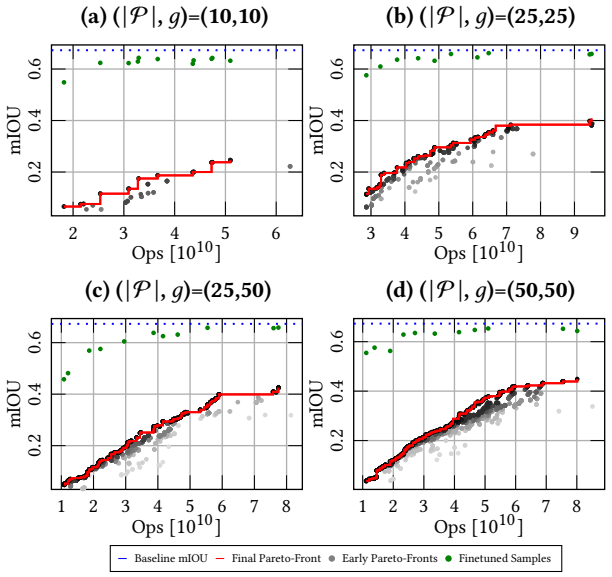

Figure 5: NSGA Search using OPs w/o compression constraints. Grey to black shades represent older to newer generations, red points belong to the final generation.

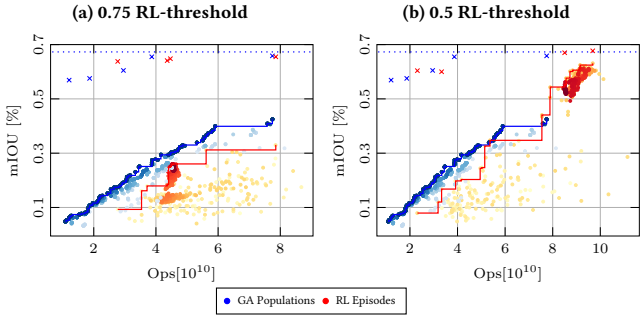

Figure 6: Comparison of our proposed GA-based channel pruning with RL search [13]. Our approach generates dominating Pareto optimal configurations compared to RL search. Crossed points are fine-tuned Pareto-optimal samples.

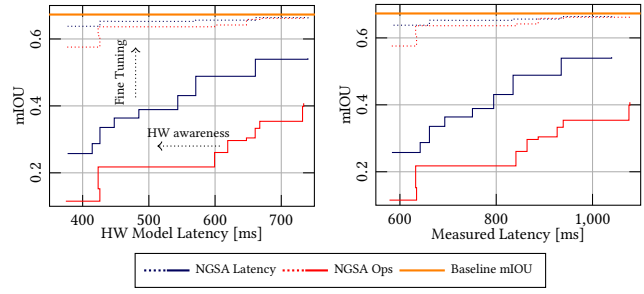

Figure 7: NSGA Search using HW Model Latency with compression constraints.

Fig. 7 illustrates the dominating pruning configurations for the two GA search scenarios. The hardware model latency values are shown on the left whereas measured latency values are shown on the right. The graphs on the left and the right portray the same trend, indicating the high fidelity of our hardware model to guide the GA-based pruning search. Our HW-aware pruning configurations yield a better trade-off between mIOU and latency compared to OPs-based pruning search. We fine-tune (dotted line) the pruning configurations (solid line) and perform inference of compressed DeepLabV3+ on our accelerator (Table 1, unrolling configuration 16, 32, 4). We observe a latency of **0.67s** (2.44× lower than baseline) and a mIOU of 65.29% (-1.98 pp). Analyzing the layer-wise compression ratios, we observe that the HW-aware pruning removes more filters in dilated convolutions compared to OPs-based pruning configurations. This can be attributed to the higher latency caused by DRAM accesses. Compared to OPs-based pruning, HW-aware pruning achieves 1.3× lower latency with similar mIOU.

5 CONCLUSION

In this work, the problem of accelerating a state-of-the-art semantic segmentation network is successfully tackled by an FPGA-based accelerator exploiting a HW-aware channel pruning technique. We improve the DRAM accesses and latency of the accelerator by exploiting loop tiling and unrolling techniques. Our baseline accelerator (before channel pruning) gives 183.3 GOPS throughput. We proposed a novel framework for automated channel pruning using GA search. We incorporated a theoretical HW model simulating the latency of the FPGA accelerator during the GA search. Using the HW-aware channel pruning, we obtain 2.44× lower latency with minimal degradation in semantic prediction quality.

ACKNOWLEDGMENT

This work is funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) through the grant 19A19013B in the project KI-DeltaLearning.

REFERENCES

[1] K. He, X. Zhang, S. Ren, *et al.*, "Deep Residual Learning for Image Recognition," CVPR, 2016.
[2] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," arXiv:1904.07850, 2019.
[3] L.-C. Chen, Y. Zhu, G. Papandreou, *et al.*, "Encoder-decoder with atrous separable convolution for semantic image segmentation," ECCV, 2018.
[4] Aydonat, Utku and O'Connell, Shane and Capalija, Davor and others, "An OpenCL™ Deep Learning Accelerator on Arria 10," in *FPGA*, 2017.
[5] S. Zhang, Z. Du, L. Zhang, *et al.*, "Cambricon-x: An accelerator for sparse neural networks," MICRO, 2016.
[6] C. Zhang, Z. Fang, P. Zhou, *et al.*, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," ICCAD, 2016.
[7] D. Wang, K. Xu, and D. Jiang, "PipeCNN: An OpenCL-based open-source FPGA accelerator for convolution neural networks," ICFPT, 2017.
[8] Y. Ma, Y. Cao, V. Sarma, *et al.*, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2018.
[9] L. Bai, Y. Lyu, and X. Huang, "A unified hardware architecture for convolutions and deconvolutions in cnn," ISCAS, 2020.
[10] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017.
[11] S. Han, J. Pool, J. Tran, *et al.*, "Learning both weights and connections for efficient neural networks," NeurIPS, 2015.
[12] Y. He, X. Zhang, and J. Sun, "Channel Pruning for Accelerating Very Deep Neural Networks," ICCV, 2017.
[13] Y. He, J. Lin, Z. Liu, *et al.*, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices," ECCV, 2018.
[14] C. Yang, Z. An, C. Li, *et al.*, "Multi-objective pruning for cnns using genetic algorithm," arXiv:1906.00399, 2019.
[15] Y. Hu, S. Sun, J. Li, *et al.*, "A novel channel pruning method for deep neural network compression," arXiv:1805.11394, 2018.
[16] T. Wang, K. Wang, H. Cai, *et al.*, "Apq: Joint search for network architecture, pruning and quantization policy," CVPR, 2020.
[17] M. Cordts, M. Omran, S. Ramos, *et al.*, "The cityscapes dataset for semantic urban scene understanding," CVPR, 2016.