M.S. THESIS

Adaptive Network Compression for DNN-FPGA Accelerator Using Layer-Sensitivity

심층 신경망 FPGA 가속기를 위한 레이어 감도에 따른 적응형 네트워크 압축 기법

August 2020

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

고지웅

M.S. THESIS

Adaptive Network Compression for DNN-FPGA Accelerator Using Layer-Sensitivity

심층 신경망 FPGA 가속기를 위한 레이어 감도에 따른 적응형 네트워크 압축 기법

August 2020

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

고지웅

Adaptive Network Compression for DNN-FPGA Accelerator Using Layer-Sensitivity

심층 신경망 FPGA 가속기를 위한 레이어 감도에 따른 적응형 네트워크 압축 기법

지도교수 Bernhard Egger

이 논문을 공학석사 학위논문으로 제출함

2020 년 06 월

서울대학교 대학원

컴퓨터 공학부

고지웅

고지웅의 공학석사 학위논문을 인준함

2020 년 06 월

위 원 장	박근수	(인)
부위원장	Bernhard Egger	(인)
위 원	Srinivasa Rao Satti	(인)

Abstract

Deep neural network (DNN) accelerators based on systolic arrays have been shown to achieve a high throughput at a low energy consumption. The regular architecture of the systolic array, however, makes it difficult to effectively apply network pruning and compression; two important optimization techniques that can significantly reduce the computational complexity and the storage requirements of a network. This work presents AIX, an FPGA-based high-speed accelerator for DNN inference, and explores effective methods for pruning systolic arrays. The techniques consider the execution model of the AIX and prune the individual convolutional layers of a network in fixed sized blocks that not only reduce the weights of the network but also translate directly into a reduction of the execution time of a convolutional neural network (CNN) on the AIX. Applied to representative CNNs such as YOLOv1, YOLOv2 and Tiny-YOLOv2, the presented techniques achieve state-of-the-art compression ratios and are able to reduce interference latency by a factor of two at a minimal loss of accuracy.

Keywords: Convolutional Neural Networks, FPGA Accelerator, Systolic Array, Network compression, Architecture-Specific Network Pruning

Student Number: 2018-29331

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

The progress of Deep Neural Networks (DNNs), in particular Convolutional Neural Networks (CNN), in image classification [29] and object detection [19] tasks has lead to a large body of published work in academia and industry alike. While GPGPUs are still widely used in the training process, the inference task is dominated by more performant and energy-efficient accelerators implemented on ASICs [8] or FPGAs [12]. Many of the proposed accelerators are tailored towards specific classes of CNNs, however, in a commercial Infrastructure-as-a-Service (IaaS) environment, the ability to utilize the infrastructure for a wide range of services is an important goal.

The inference accelerator presented in this paper is based on the previously developed [7] and has been explicitly designed to support arbitrary forms of neural networks including CNNs and recurrent neural networks (RNNs) on both ASICs and FPGAs. The core of presented accelerator is a systolic array executing matrix and convolution operations. On the lowest level, a workload is partitioned into blocks of 8-byte depth, simultaneously processing 4 and 8 input

channels for 16 and 8 bit data, respectively. Many optimization techniques for FPGA has been proposed to increase energy efficiency and throughput of DNNs. These include matrix decomposition [10, 31], network quantization [38, 13] and weight pruning [33, 21], etc. We focus on weight pruning technique, since the proposed architecture supports indirect addressing mode for pruned weights and the main overhead is computational operations not from the memory load.

Different techniques from pruning individual weights (fine grained) to entire filters (coarse-grained) have been proposed and evaluated [24], leading to the intuitive conclusion that finer-grained pruning is able to prune more weights for a given accuracy. However, the execution units of the state-of-the-art hardware exploit filter or channel-level granularity of the sparsity, so the weights pruned by the first contribution has limit to accelerate inference time on the FPGAs. Most recent works has been focus on the second contribution, which can both fully exploit hardware/software library and improve the performance of specialized FPGAs for the pruned weights. These works also have limit to reduce computational cost in our hardware that the granularity of the execution unit is 4 or 8 input channels.

In this paper, we proposed architecture-specific pruning scratch, which is tailored pruning method that exploits the block processing structure of the inference accelerator. Our approach repeatedly prunes blocks which grouped 4 or 8 input channels from the networks, compresses the networks, then fine-tunes the compressed model. The technique for the selection of the pruned layer and channels have been explored. This method computes the sensitivity of each layer by computing the accuracy of each layer pruned by one block comprising the 4/8 channels with the smallest average weight. Network compression is applied to reduce weight parameters by pruning all filters at identical input channel indices. Fine-tune is performed one epoch at a time and up to 20 epochs;

the parameters with the best mean Average Precision (mAP) are selected and pruned in following next pruning steps. We experimented our scratch to object detection networks such as Tiny-YOLOv2, YOLOv2 and YOLOv1, achieved 3.6x reduction in computing operations and 12.9x parameter compression of the Tiny-YOLOv2 pre-trained weight despite its coarse-grainedness.

Chapter 2

Background

2.1 Object Detection

Object detection is applied to one image but detects and classifies several objects per image. The task of object detection thus includes displaying the location of the objects as rectangular shapes. Image classification and object detection share a lot of common points as they are similar problems. Traditionally, image classification is based on convolutional neural networks, and object detection relies on this same type of networks. However, object detection is more complicated than image classification as it has to detect, classify, and localize all the different entities in an image. Thus, the processing time to achieve an acceptable level of accuracy is much higher.

2.1.1 mean Average Precision (mAP)

The AP metric used in object classification benchmarks. There is an AP value for each class, and the average of this is called mAP. It is composed of two parts:

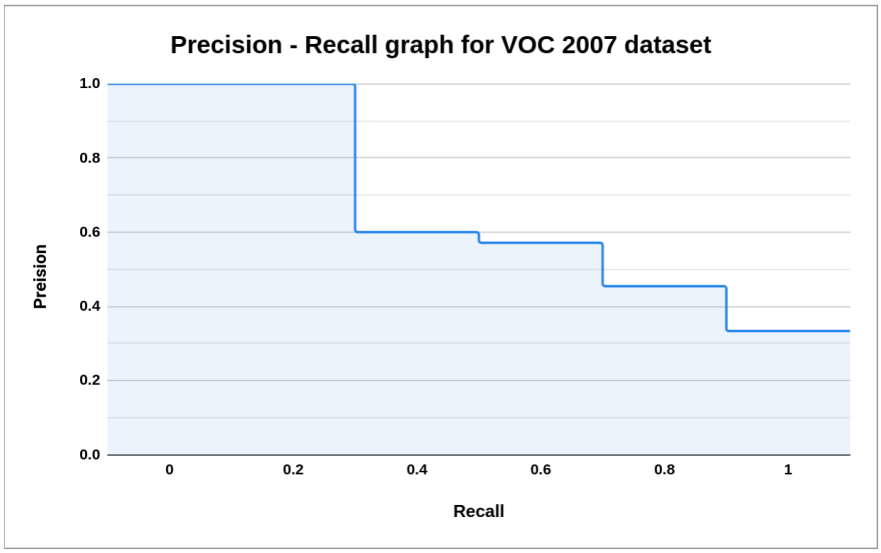

Figure 2.1: PR-curve examples from one of the class in the PASCAL VOC 2007 dataset [11]. The blue area represents AP.

precision and recall. Precision P, also known as the positive predicted value, is the ratio of true positives (TP) over the total number of predicted positives (true positives plus false positives (FP)), i.e., The recall R, on the other hand, also known as the true positive rate of a given class, is defined as the ratio of true positives over the total of ground truth positives (true positives plus false negatives(FN)), i.e. Precision and recall can be used to create an RP-graph as shown in Fig. 2.1, and the AP is the size of this area. In the VOC 2007 dataset, recall is divided by 11 points, increased by 0.1 from 0.0 to 1.0, and the precision value corresponding to that point is taken.

Intersection over Union (IoU), measures how much the predicted bound box overlaps with the one of the ground truth of a certain object. It is used to determine if a detection is correct or not. The correctness threshold can be set to different IoU ratios. For the COCO dataset [22], the AP is averaged for IoUs

from 0.5 to 0.95. That's where mAP comes from. On the other hands, VOC dataset uses IOU over 0.5. If a value is given after mAP (like mAP-50) it states which IoU was used as correctness threshold.

2.1.2 YOLOv2

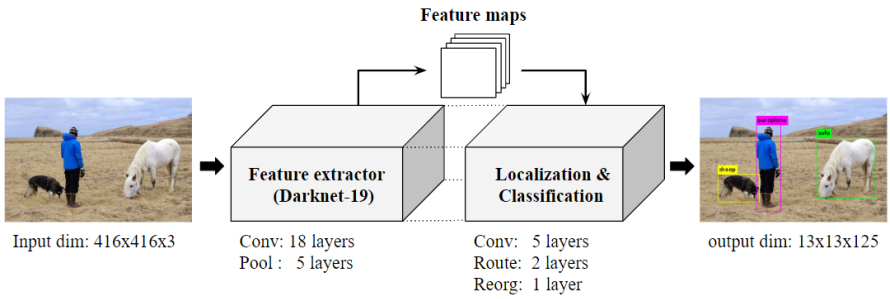

Figure 2.2: The overview of YOLOv2 structure and the process of inference using VOC PASCAL image.

You only look once (YOLO) is a model for the real-time object detection. The second generation of YOLO was released on December 25, 2016. YOLOv2 [19] improves YOLOv1 [26] which is suffered from localization error and low recall rate with a number of tricks and is more capable of detecting objects faster. YOLOv2 extracts features of input images, then predict coordinates of bounding box and class probabilities at the same time. The YOLOv2 network architecture can be divided into two parts. The first part is composed of a feature extractor called Darknet-19 [25]. Darknet-19 itself is composed of 19 convolutional layers featuring batch normalization and leaky ReLU activation. Darknet-19 also contains 5 max pooling layers to down-sample the feature map. The second part of the network is responsible for detection. This part is attached to the back, after the convolutional layer of Darknet-19 is removed. The detection module is composed of several convolutional layers as well as

route and reorganisation layers that allow the detection to better benefit from features extracted earlier in the network.

The input images are down-sampled by 2×2 pooling operation with a stride of 2, and finally divided into grids of $S \times S$, and each grid has B bounding box information. Each bounding box contains confidence scores and four coordinate x, y, w, h. The confidence score indicates probability of containing object in prediction box. It is defined as $Pr(Obj) \times IOU_{pred}^{truth}$, which $Pr(obj)$ denotes prediction probability, and IOU_{pred}^{truth} means IoU between prediction box and the ground truth. Each grid cell also contains C conditional class probabilites, calculated as following:

$$Pr(Class_i|Objdp) \cdot Pr(Obj) \cdot IOU_{pred}^{truth} = Pr(Class_i) \cdot IOU_{pred}^{truth}$$

In summary, each grid cell of $S \times S$ contains $(B \times (1+4)+C)$ information. VOC PASCAL dataset uses $S = 13$, $B = 5$, and $C = 19$, so the final output dimension of YOLOv2 is $13 \times 13 \times 125$. As shown in Figure 2.2, YOLOv2 contains routing layers which forward layers from earlier parts in the network towards the end, and reorganization layer which transforms the bigger input size into additional channels. We will discuss the pruning method in detail later on for these two layers.

2.2 AIX Accelerator

2.2.1 Overview of AIX Architecture

AIX is a dataflow machine for accelerating neural networks. All the neural networks are represented by data flow graphs with nodes of layerwise operations such as convolution and perceptron. Similarly, as shown in Figure 2.3, AIX has several HW blocks, each of which corresponds to one or more of the operations. For example, matrix multiplication and convolution (MXC) block processes 2D

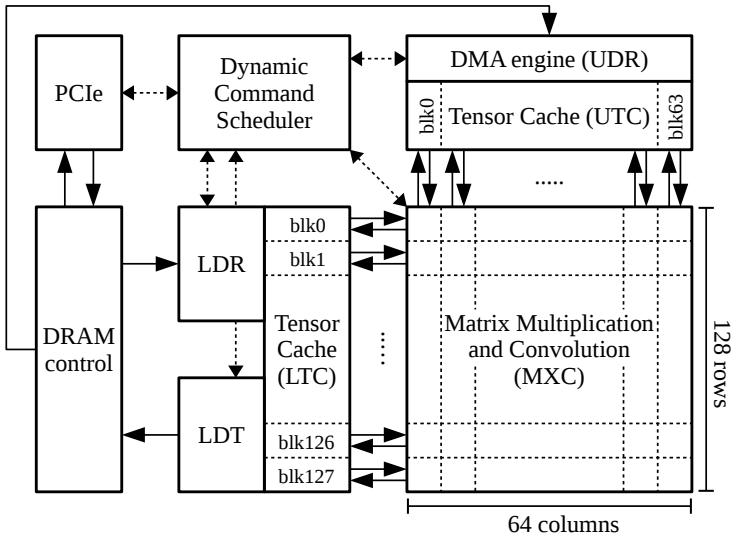

Figure 2.3: High-level architecture of the AIX accelerator. blk denotes blocks in tensor caches, which divide the tensor caches by the same number of rows and columns for each 128×64 MXC processor.

convolution or perceptron layer in a neural network. Also, feature map concatenation run on direct memory access (DMA) engines such as LDR. Running a neural network on AIX, therefore, means finding a good matching between layer-wise operations in a neural network and HW blocks on AIX, and then scheduling the operations with the minimum latency as possible.

In order to facilitate those executions of commands without stall, AIX has a memory hierarchy consisting of two different scratch pad memories and DRAM. The scratch pad memories are called tensor cache (TC). Basically DMA engines fills TCs with data newly fetched from DRAM while the previously fetched data is consumed by the HW blocks in AIX. There are two different types of tensor caches: left tensor cache (LTC) and upper tensor cache (UTC). UTC is a read only scratch pad memory in the stand of computing HW blocks as UTC

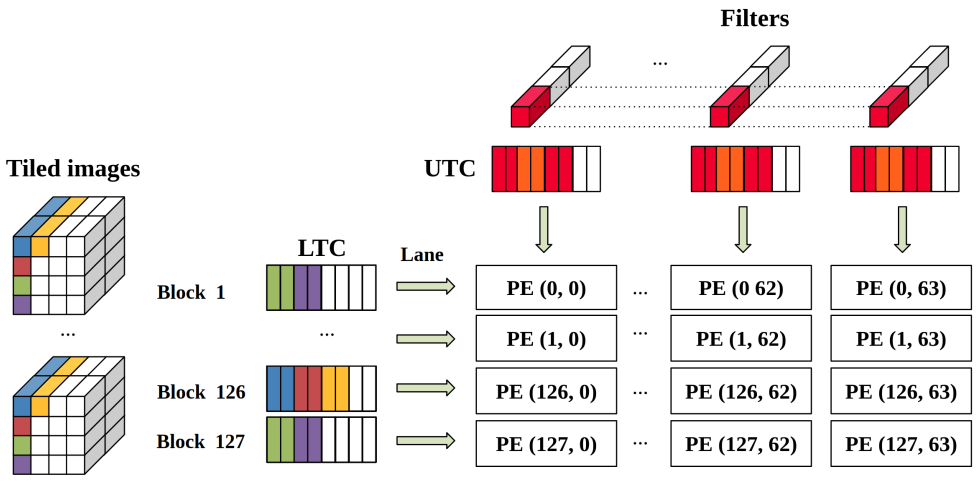

Figure 2.4: The MXC processes workloads in blocks. The input feature maps are tiled 64 bits size of consecutive channels, and loaded to LTC blocks. The tiled values in each LTC blocks are assigned to the corresponding MXC rows.

mainly stores weights and biases. This reflects that the weights and biases in a neural network has read-only property for inference as they are frozen after thorough training. On the other hand, LTC mainly provides input data and receives the computation results from other HW blocks in AIX. There are three DMA engines: LDR, LDT, and UDR. As explained, UDR fills the read-only UTC. When LDR fills LTC, LDT simultaneously writes the output feature maps from LTC to DRAM.

2.2.2 Dataflow of AIX Architecture

We will now look a bit closer at how the MXC unit computes convolution as it explains the pruning approach we adopted. The LTC has 8192 entries of 8192 bits which gives us a size of 8MB. Every entry is divided into 128 64-bit blocks. Each MXC row is mapped directly to one block in the LTC and connected

by individual lanes. In a similar fashion, the UTC has 6144 (=6k) entries of 4096 bits divided into 64 64-bit blocks, thus a total size of 3MB. The MXC unit is a two dimensional systolic array composed of 128 rows and 64 columns where each node operates on an input block (fed from the LTC) and a weight block (fed from the UTC). In other words, blocks are the atomic elements MXC performs operations on. To compute an output pixel of a convolution, we need to multiply the input pixels with those coordinates (and their neighbours for filter sizes above 1×1) from all the input channels with all the weights of a filter (channel by channel), then add up the individual products. The dimension we navigate along the most is the one of channels. Therefore, we load the input values in NHWC format instead of the usual NCHW (N: batch, H: heigth, W: width, C: channels). Channels being loaded consecutively in memory and blocks being 64 bits long, a block contains 8 (resp. 4) channels when the data size of a value is 8 (resp. 16) bits. In Fig.2.4, we can see in what order blocks are loaded into to the LTC and UTC (and therefore fed into the MXC). In the UTC, each column contains the weights of an individual filter, channels being consecutive here too. When executing the systolic array, the PEs on the top receive the data from the UTC and the PEs on the left from the LTC. The weights are then propagated from top to bottom and the input values from left to right. Each PE contains the partial sum computed so far. Once all the data of a filter and the corresponding input is processed, the PE's partial sum will contain the value of a specific output pixel. This value is saved in the PE's output register and propagated left in order to be saved back into the LTC and later into memory. The weights are fed in a circular manner so when one output pixel has been computed, we set the partial sum of the PE to 0 and we can immediately process the next pixel receiving the filter's weights from the beginning again and the next required input data. Once all data from the

caches processed, a part of the LTC dedicated to the output will contain the values of the computed output feature map, ready to be saved in memory.

Chapter 3

Implementation of Pruning on AIX Accelerator

3.1 Convolutional Neural Network (CNN)

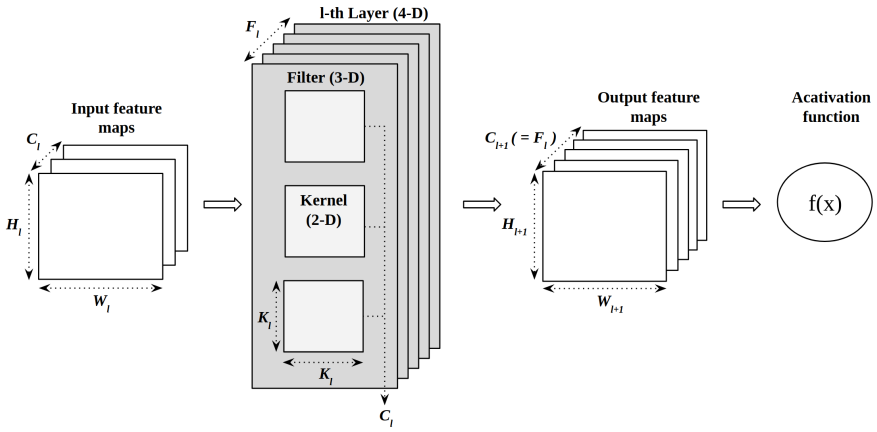

Figure 3.1: The process of convolutional neural network in l-th layer and notation of weights.

A convolutional neural network (CNN) is a deep neural network used to

classify or detect visual images. The network extracts features of images by convolving input data with weights, and then passing activation function such as ReLU or Sigmoid. The outputs of activation function feed forward to next convolutional layer. Weights consist of 4-dimensional tensor in l-th layer, $w \in R^{C_l \times K_l \times K_1 \times F_1}$, where F_l, C_l denotes total number of filters and input channels, and K_l is the spatial size of the 2-D kernel in the l convolutional layer. After convolving operation, it produces output feature-map $H_{l+1} \times W_{l+1} \times F_l$ which is input feature map of $l+1$ convolutional layer. The output feature map becomes the input feature map of the next layer. The notations and process of CNN in l-th layer are shown in Figure 3.1 .

3.2 Granularity of Sparsity for Pruning CNNs

Pruning is a technique that consists in removing certain weights of a CNN to speed up its processing time while maintaining some degree of accuracy. We follow the basic idea presented in [6]. The paper offers an overview of this approach and introduces a total of four ways to prune convolutional layers.

Scalar (0-D) pruning, delete individual weights is the most fine-grained and also the most irregular sparsity. Vector-level (1-D), kernel-level (2-D), and filter-level (3-D) sparsity represents more regular and coarse-grained pruning methods. Fine-grained pruning, each weights are deleted based on their absolute value. This leads to highly irregular pruning patterns with zero values distributed throughout all filters. While a number of accelerators support zero-value skipping to reduce the energy consumption and/or the computation time, the weight matrices cannot be easily compressed and keeping all processing elements busy in the presence of zero values is still a challenge. The coarser-grained pruning methods remove adjacent weight, leading to a more regular pattern of

sparsity. To decide which values are pruned in each granularity, we adopt the L1 norm is adopted as proposed in [7]. The L1 norm serves as a simple magnitude-based pruning criterion and is computed by

$$S_i = \sum_{W \in G_i} |w|$$

G_i and S_i denotes the group of grain i that comprise of multiple weights, and sum of absolute values in the corresponding grain respectively. Pruning is performed by first sorting in ascending order based on S_i, then the lowest x percent are removed. To experiment the effect of accuracy drop on grain size, we implemented four different granularity of sparsity in YOLOv3 [27]. We extract all the weights (gathered together in the desired regularity), sort them, and set the smallest values to 0 according to the given sparsity. We prune 10% of S_i

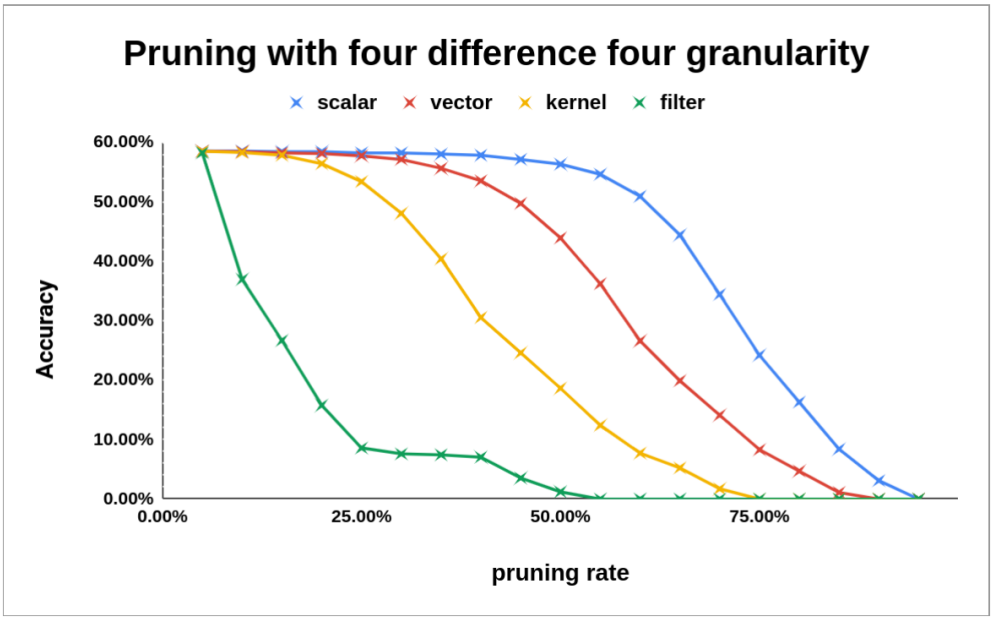

Figure 3.2: The accuracy drops according to pruning ratio for four different granularity of sparsity.

and record accuracy iteratively until all the weights are removed.

In Figure 3.2, we can see that the coarser the introduced sparsity, the stronger the impact on the networks detection performance. This is an expected result as coarse-grained pruning removes larger chunks of information and also does not necessarily only prune the smallest values but rather structures with the lowest average values. While vector-level pruning noticeably starts dropping at around 35% of sparsity and then maintains an almost constant relationship of accuracy compared to scalar pruning, the accuracy of kernel-level pruning starts dropping at 20% and is significantly lower than pruning at vector or scalar level. Filter-level sparsity has a catastrophic impact very quickly. Despite the significant drop, recent research adopt filter-level sparsity to exploit MKL sparse BLAS or cuSPARSE library, and fine-tune helps recover this damage.

3.3 Network Compression for Channel Pruning

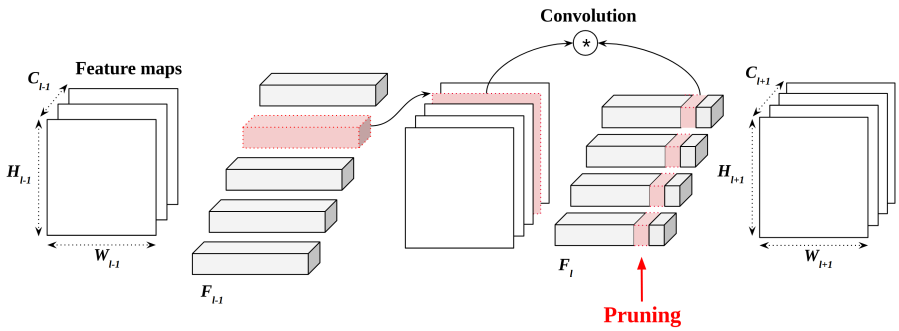

Figure 3.3: The process of network compression for pruned-input channel (red color) in l-th convolutional layer.

The input channels being pruned, we could eliminate connection with other weight parameters by setting the pruned values to zero. To preserve the networks consistency and to save further computation, we also eliminated the filters that generate these pruned channels from the previous layer. Fig. 3.3 shows the

fourth input channel of l-th layer is pruned. The fourth input feature map of l-th layer is no longer requires convolution operation, since the input channel is removed. Accordingly, we eliminated together with the filter in F_{l-1}, that generated fourth input feature map in the previous layer. In addition to removing the weights and other parameters corresponding to the compressed channels and filters in the network's weight file, we also updated its configuration file were the number of filters per layer are saved.

In summary, the procedure of compression is the following:

- Identify and memorize the pruned input channel(s)
- Mark and memorize the corresponding filters to be compressed
- Eliminate input channel(s) and filter(s) from the weight file
- Update the configuration file to represent the compressed network

3.4 CNN Pruning on AIX Accelerator

The execution mode of the accelerator dictates that only removing an entire block leads to an reduced execution time in the current layer. In this sense, we proposed accelerator-specific block-granularity pruning which grouped four or eight input channels pruned at once instead of single input channel or filter.

3.4.1 Block-Granularity for Pruning

A block is an 8-bytes buffer with the (height, width, input channels) dimensions $1 \times 1 \times 8$ or $1 \times 1 \times 4$ depending on the element's data size (8/16 bits). We gather channels from each layer into groups of 4 or 8 depending on the data size. We want to prune blocks having the less influence on the output so we gather channels according to the average magnitude of their weights and

not according to spatial consecutiveness. The reason channels in a same block don't need to be consecutive is that they will be completely removed during compression which we will discuss in more details later. The exact procedure to form blocks in each convolutional layer l is as follows:

1. For each channel in layer l, compute the l_1-norm of their weights to evaluate their importance. The following equation is used:

$$s_l^c = \frac{\sum_{i=1}^{F_l} \sum_{j=1}^{K_l} \sum_{k=1}^{K_l} |w_{icjk}|}{F_l \times K_l^2}, \{1 \leq c \leq C_l\}$$

2. Sort the channels in ascending order according to their computed s_l^c.

3. Gather the four or eight channels with the smallest s_l^c from the sorted list into a block based on the element's data width (8/16bits). The next block will be composed of the next smallest channels, and so on until all channels of the layer are gathered into blocks.

4. Calculate B_l^k, $\{1 \leq k \leq *\frac{C_l}{4}\ or\ 1 \leq k \leq *\frac{C_l}{8}\}$, the l_1-norm of the blocks' weights (computed from the s_l^c of the channels they are composed of).

We create the blocks from all convolutional layers in the network, and store the obtained blocks' information in a structure array I. The block information includes: layer id, index of the channels composing the block and B_l^k. In recent object detection networks [19, 28], the number of channels in each convolutional layer are usually divisible by four or eight. However, if it is not divisible or the number of input channel size is smaller than the size of a block, the remaining channels channels form a block.

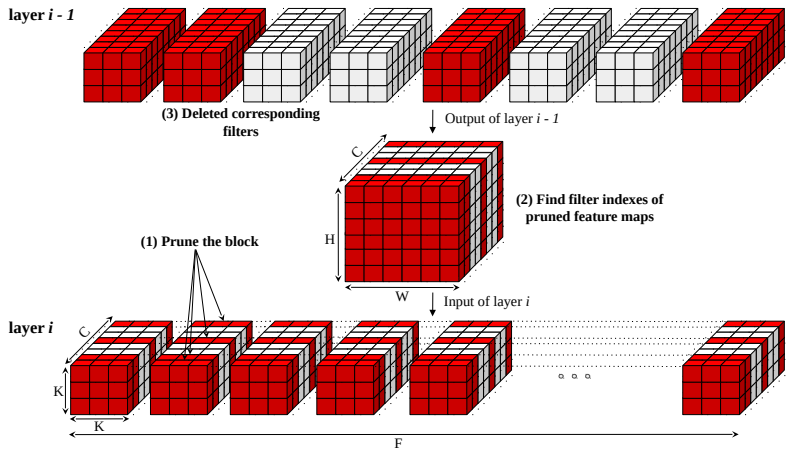

Figure 3.4: Network compression for block granularity. The figure describes prune input channel index i, and remove corresponding outpuit filters.

3.4.2 Network Compression for Block Pruning

AIX accelerator does not support zero-skipping, so we have to implement compression step for block granularity. Four or eight input channels that consist single block should not be contiguous. Network compression helps to compact pruned input channels which have the effect of removing the entire block. The Fig.3.4 shows that the corresponding filters are deleted in the $i-1$ layer when a block consisting of 4 input channels in the i layer is pruned (16-bit width). If compression is performed after deleting the red part in layer i, four input channels equal to the size of the block are removed. The gray input channels then become neighbors. We can carefully prune input channels individually through compression, which is more fine-grain way compared to contiguous configuration.

Chapter 4

Adaptive Layer Sensitivity Pruning

4.1 Overview

The whole process of the scratch describes in Fig.4.1. *pruning i step(s)* is the procedure from pruning the blocks to save best epoch after fine-tune. At first, We group block-level sparsity when the pre-trained weights comes in. Then, prune blocks based on layer-sensitvity pruning algorithms which would be discussed later. The pruned input channels filled with zeros are removed through network compression and the neural network structure is changed. We fine-tune the N number of epochs to improve the accuracy of the modified neural network, then select and update the weights with the highest accuracy recovery among the N. We called this method fine-tune method 'keep the best epoch', and we'll talk it later. The stop condition of this scratch is when the accuracy recovery does not occur even after learning further.

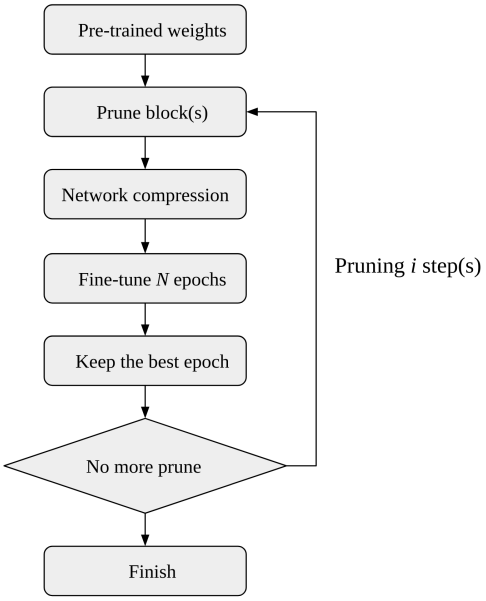

Figure 4.1: The whole process of architecture specific pruning

4.2 Layer Sensitivity Graph

Existing coarse-grained pruning methods process pruning sequentially layer by layer [6, 16, 18], which only consider local information. However, this approach could drop the accuracy significantly, since some layers play important roles in the network [21]. Some researchers [15, 37] show pruned object detection network by transferring pruned pre-trained weight of classification network to the feature extraction part which cannot reduce input channels of detection parts.

Inspired by [39], each step, we evaluated the sensitivity of each convolutional layer regarding accuracy. For the experiment, we removed one block from each convolutional layer of YOLOv2 and measured the accuracy using a COCO dataset. The stop condition is when all blocks are removed in the layer or the

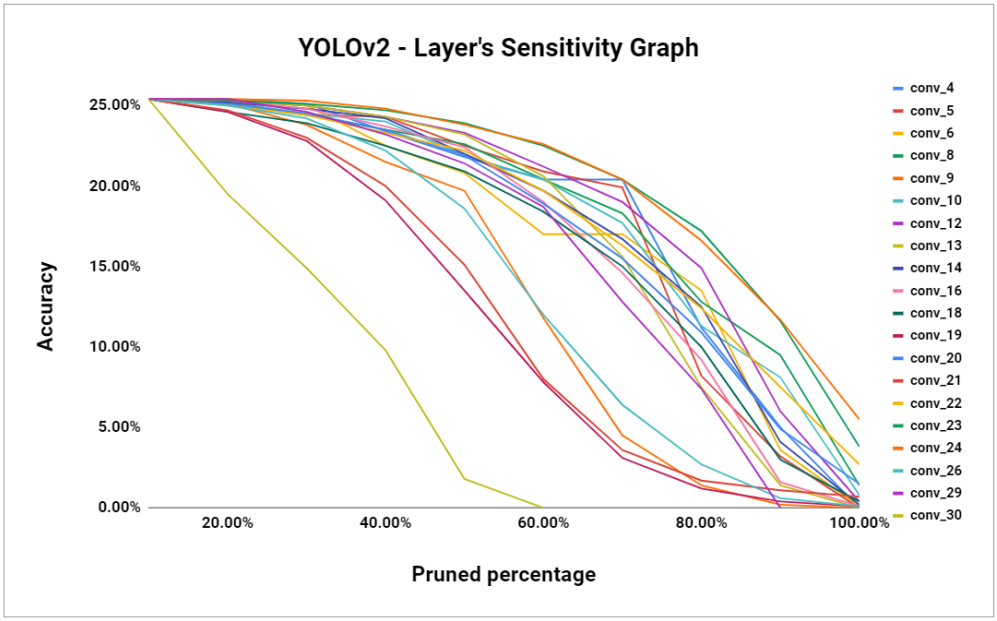

Figure 4.2: Layer sensitivity graph of YOLOv2 using COCO dataset.

accuracy is zero. We make the graph in Fig 4.2, and called *layer sensitivity graph*. The 0-th and 2-nd layers of YOLOv2 are also convolutional layers, but the size of the input channel is small (3 and 32 respectively). It is not possible to make enough blocks, so they are not added to the graph. As shown in Fig 4.2, the 20-th (orange line) and 21-st (green line) layers show the smoothest drop. Both layers are one of the five convolutional added to Darknet-19 classification network, and has 1024 input channels per filter. This is the second largest number in YOLOv2, and there are no changes in dimension or addition of input feature maps. As a result of that, two convolutional layers has redundant input feature maps. On the other hands, layers 29 (red line) and 30 (olive line) show a steep drop. The 29-th layer receives merged feature maps of two layers with different resolutions. one of them is reorganizataion layer we will talk it later. 30-th layer is the last convolutional layer which has output tensor of the

final outputs ($(B \times (1+4)+C)$) in YOLOv2. Pruning this layer means loss of information in the final output, so it directly affects the accuracy.

4.3 Concept of Adaptive Layer Sensitivity Pruning Algorithm

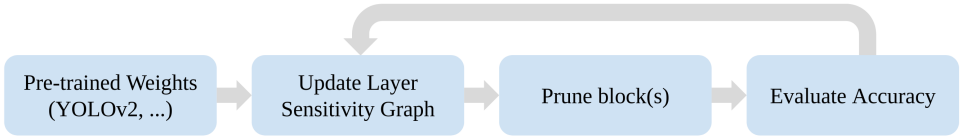

Figure 4.3: Algorithm description of adaptive layer sensitive pruning

We proposed the pruning algorithm based on the layer sensitivity graph described in section 4.2 and called it adaptive layer sensitivity (ALS) pruning. The reason for we added 'adaptive' was that the pruning algorithm automatically decide how many blocks were going to removed depending on the size and characteristics of the pre-trained weight. Our pruning algorithm implemented four stage process, as shown in Fig. 4.3. It starts from feeding pre-trained weights which completed learning process by conventional network training. Making layer sensitivity graph by pre-trained weight is the next step. The third step is to select the one with the smallest drop among the convolutional layers and then remove the block to update the weight. Removing one block at a time can prevent a significant drop, but we added a function so that the user can determine the number of blocks that can be removed at one time, since it can take a long time depending on the capacity of the accelerator. The final step is to evaluate this pruned weights, if the accuracy drop is higher than the threshold we set, move to the next process; fine-tune. Otherwise, network pruning is done iteratively by updating new layer sensitivity graph.

4.4 Discussion on Adaptive Layer Sensitivity Pruning Algorithm

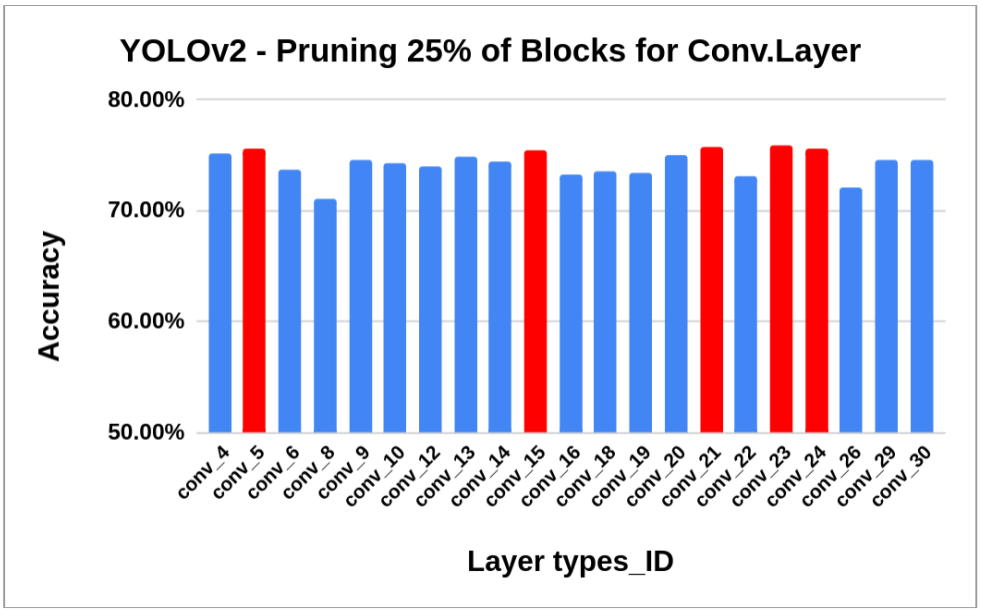

Figure 4.4: Pruned 25% of each convolutional layer individually in YOLOv2 and evaluate accuracy by PASCAL VOC 2007 test datset. The red bars indicate the layer that the drop of accuracy less than 0.5%.

ALS pruning removes one convolutional layer's block(s) per iteration, but we can also devise a method to select multiple convolutional layers with less drop in the layer sensitivity graph. However, we found through experiments that this method could lead to a significant reduction in accuracy. For the experiments, we pruned 25% of blocks for each convolutional layers independently in YOLOv2 and evaluate the accuracy by VOC PASCAL test dataset, and results are shown in Fig. 4.4. The original accuracy of YOLOv2 on PASCAL 2007 test dataset is 75.87%. Among the convolutional layers, Among the convolutional

layers, four layers with accuracy drop of less than 0.5% was selected and displayed in red on the graph. We removed these layers at the same time, and then measured the accuracy. As a result, it showed a drop of 15.23% accuracy, and found that when pruning, all the layers are influenced by each other. Therefore, when using a layer sensitive graph, it is more effective to select one layer and prune the block in it.

4.5 Compression for YOLOv2 multi-branches

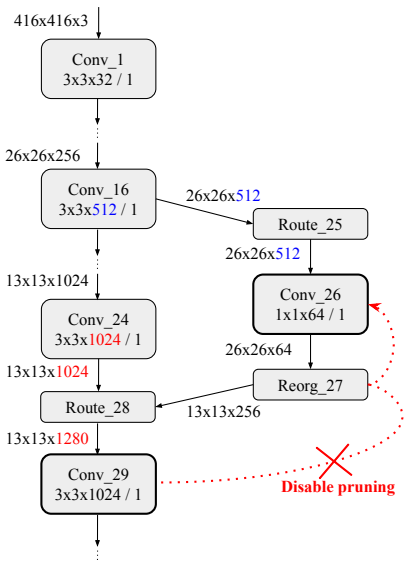

Figure 4.5: YOLOv2 compression method for the two route layers. The blue and red values are affected factors when the first and second route layers are compressed, respectively.

The compression technique directly applicable to single branch object detection networks such as Tiny-YOLOv2, YOLOv1. However, the networks with multi-branches such as YOLOv2 needs to be handle, since the index of the pruned one input channel does not correspond to only one output channel. In

this paper, we will focus on handle route layers in YOLOv2.

YOLOv2 contains routing layers which concatenate the feature maps of the layers from earlier parts in the network towards the end. When we mark the filters that generate channels to be pruned, there are usually located one layer (or two when there are maxpool layers) above the current convolutional layer. Route layers create links with layers further away, so we should consider the connection among the layers when pruning blocks. Fig.4.5 shows there are two route layers in the YOLOv2, with respective indexes of 25 (Route_25) and 28 (Route_28), since we indexed all layers of YOLOv2 in order based on the main branch. We differentiate these two cases to handle the pruning of route layers in the YOLOv2.

- Route_25: This route layer forwards the output of Conv_16 to the input feature maps ($26 \times 26 \times 512$) of Conv_26. Filters in convolutional layer 16 are affected by pruned channels in both convolutional layer 18 and 26. Indeed, we mark and compress filters that generate pruned channels in either layer 18 or 26. So the same channels will be pruned in both layer 18 and 26. In other words, pruned channels in layer 18 will also be pruned in layer 26 and vice versa.

- Route_28: this route layer merges the output of the network's main branch and the forwarded branch. However, we only apply the compression to the part that hasn't been routed (main branch). This is the case because the routed branch has the bigger feature map size from layer 16 (26×26 instead of 19×19 in the main branch) so the output of layer 26 goes through a reorganization layer (layer id: 27). This layer transforms the bigger input size into additional channels. So a channel in the front part

of layer 29 (see figure below) does not have an one-on-one correspondence with filters from the previous layer anymore. A filter from the previous convolution generated 4 of layer 29 channels.

4.6 Fine-tune

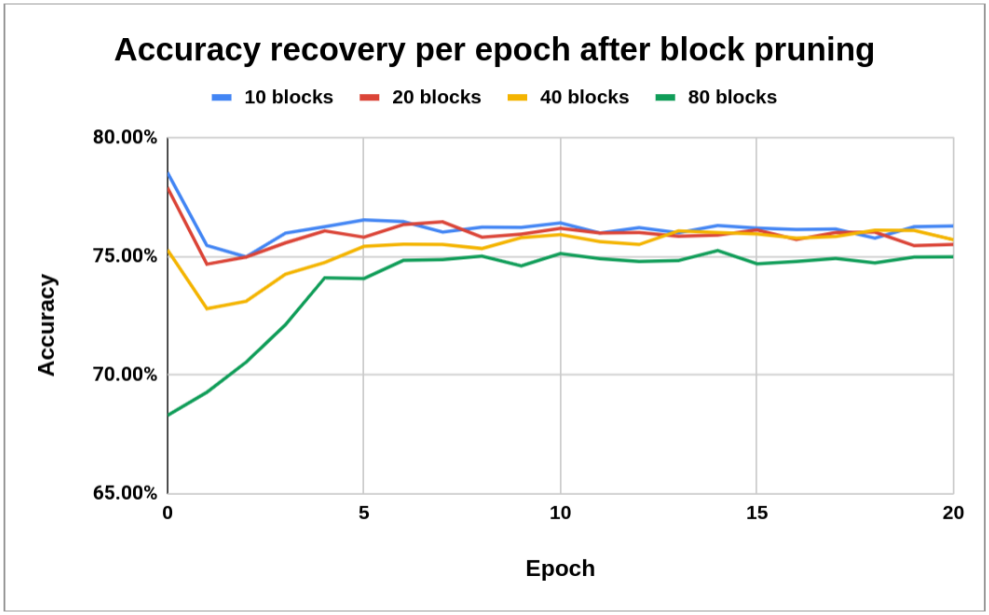

Figure 4.6: Fine-tuned 20 epochs after 10/20/40/80 blocks are pruned in YOLOv2. We measured the accuracy after learning one epoch at a time.

The accuracy drop drastically due to changes the connection between the weight parameters. Fine-tune is essential process to recover the accuracy and re-evaluate the weight connection, after network compression. [15, 21] fine-tuned a certain percentage in the original training epoch, and then updated the final result. However, this method has two problems. First over-fitting may occur, when accuracy drop is negligible after pruning. Second, the final result may

not be the best recovery accuracy, since early recovery induced saturation of accuracy. Fig. 4.6 shows the accuracy results for each epoch, when fine-tune 20 epochs after pruning 10, 20, 40, and 80 block-words of YOLOv2. There are no accuracy drops for pruning 10 and 20 blocks simultaneously. Rather, both experiments increased the accuracy of 0.42% and 0.30%, respectively. We supposed this is due to reducing redundant information. As soon as the fine-tune started, 0.35% and 1.24% drop occurred. This is because the over-fitting occurred by training without changing the neural network much. On the other hands, pruning 40 and 80 blocks suffered 0.64% and 7.6% accuracy drop respectively. However, the best accuracy points were 18 and 14 epoch rather than the final results (N=20).

Chapter 5

Experimental Setup

The GPU model used for training and evaluation was RTX 2080 Ti [4], and the CPU model was Intel(R) Xeon(R) E5-2620 [5] for inference. The AIX had been implemented on a Xilinx VU13P [3] FPGA. The MXC ran at 775 MHz, the maximum frequency of the DSPs, the other parts of the accelerator run at half that frequency. The hardware utilization was limited by the DSP core at 84%.

We evaluated object detection networks YOLOv1, YOLOv2 and Tiny-YOLOv2 on PASCAL VOC 2007 test-dataset. We used Darknet framework [25] which provided pre-trained weights of the object detection networks for our experiments. We experimented with 8 input channels in one block, since the default quantization mode of AIX is 8-bits. We pruned a block after creating layer sensitivity graph, and stopped the algorithm if the accuracy drop occurred more than 1%. Fine-tune was performed one epoch at a time and up to N epochs. During the fine-tune stage, we set the number of N to 20 in this experiment, but this number could be changed by the user, taking into account the side effect

between fine-tune time and accuracy. We retrained object detection networks on PASCAL VOC 2007+2012 train and validation set. We used same optimization setting and batch size, except the learning rate which applied 1/10 of training from scratch. The stop condition of our technique was if all the convolutional layers in the networks left one block number of input channel, or the adaptive layer sensitive pruning algorithm stops without removing a block more than 10 times.

Chapter 6

Experimental Results

6.1 Overall Results

Network	Error (ϵ) range	Method	Acc. (%)	Acc. ↓ (%)	FLOPs	FLOPs ↓ (%)	Param.	Param. ↓ (%)
Tiny-YOLOv2	$\epsilon \leq 0$	Baseline	54.06	-	6.97×10^9	-	1.58×10^7	-
		ALS	**54.37**	**-0.31**	1.94×10^9	**73.07**	1.22×10^6	**92.34**
YOLOv2	$\epsilon \leq 0$	Baseline	75.83	-	2.93×10^{10}	-	5.63×10^7	-
	$0 < \epsilon \leq 0.5$	ALS	75.38	0.45	2.19×10^{10}	20.52	4.06×10^7	27.94
	$0.5 < \epsilon \leq 1$	**ALS**	**74.83**	**1.00**	1.83×10^{10}	**37.26**	3.52×10^7	**53.47**
YOLOv1	$\epsilon \leq 0$	Baseline	67.01	-	4.01×10^{10}	-	6.01×10^7	-
	$0 < \epsilon \leq 0.5$	ALS	66.69	0.32	2.80×10^{10}	30.02	5.40×10^7	10.10
	$0.5 < \epsilon \leq 1$	**ALS**	**66.01**	**1.00**	2.31×10^{10}	**42.32**	5.16×10^7	**13.13**

Table 6.1: The comparison of the results on three object detection networks. "Error (ϵ) range" is to compare the results by dividing it into three section according to the error range caused by pruning. "Acc. ↓" column indicate that the Acc. of the baseline model minus the pruned model.

As shown in Table 6.1, we compared the results of the baseline and ALS pruning from the scratch within 1% error of the Acc. for three object detection

networks. We divided the error (ϵ) range into three parts: $\epsilon \leq 0$, $0 < \epsilon \leq 0.5$ and $0.5 < \epsilon \leq 1$, and analyzed how computational complexity and parameters were pruned. The metric of accuracy was used mAP. YOLOv2 and YOLOv1 accelerate computational operations 1.6× and 1.7×, respectively in the $0.5 < \epsilon \leq 1$. YOLOv1 pruned parameters only 13.13% of the original weights. This is because, we do not consider to prune fully-connected network for this experiment, since most of the overhead in the AIX comes from convolutional layers. The light version of YOLOv2, Tiny-YOLOv2 achieved 3.6× faster and saved 12.9× parameters compared to baseline pre-trained weights without loss accuracy ($\epsilon \leq 0$). We could not record the rest of the range, since the accuracy drops occurred more than 1.2% after further pruning.

6.2 Effect of Adaptive Layer Sensitivity Pruning

ALS automatically determines how many blocks are removed per pruning step according to the network condition. We observed how many blocks are removed according to the pruning step, in order to verify that the number of blocks actually prune was well determined. To the experiments, we pruned Tiny-YOLOv2 pre-trained weights, and evaluated PASCAL VOC 2007 test-dataset. Despite the 5 pruning steps, if there were no blocks to be pruned, We finished the experiment.

Fig. 6.2 shows that how many blocks are removed per pruning step and the reduction of BFLOPs. ALS aggressively removes more than 10 blocks until the initial 12 steps, however do not remove more than 2 blocks after the 23 steps. After 47 steps, more than 1% accuracy drop occurs when pruning the block. Through this experiment, ALS correctly detects the redundancy of initial pruning steps, and aggressively remove the blocks. The network inher-

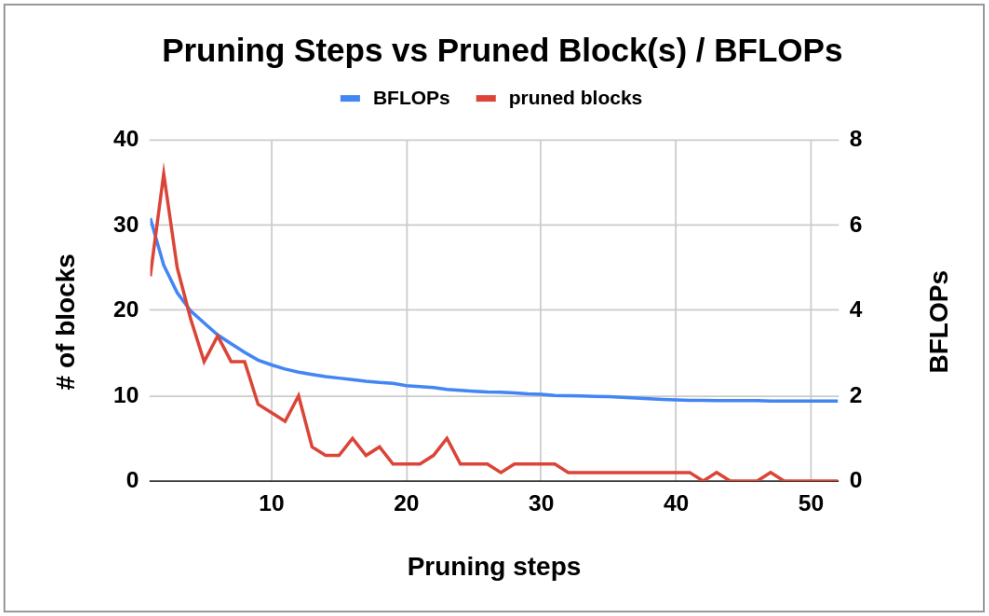

Figure 6.1: The number of blocks removed and the amount of change in BFLOPs according to the pruning step

ited by previous weight that is not enough pruned occurs two problems when fine-tuning. First, the weight might be stuck in local minimum [23], and second, performance may be degraded due to over-fitting. ALS prevents those two problems, as it processes until redundant blocks are all eliminated. In addition, We found that ALS automatically reduced the number of blocks to be pruned and fine-tune frequently if the remaining blocks have a great impact on the accuracy.

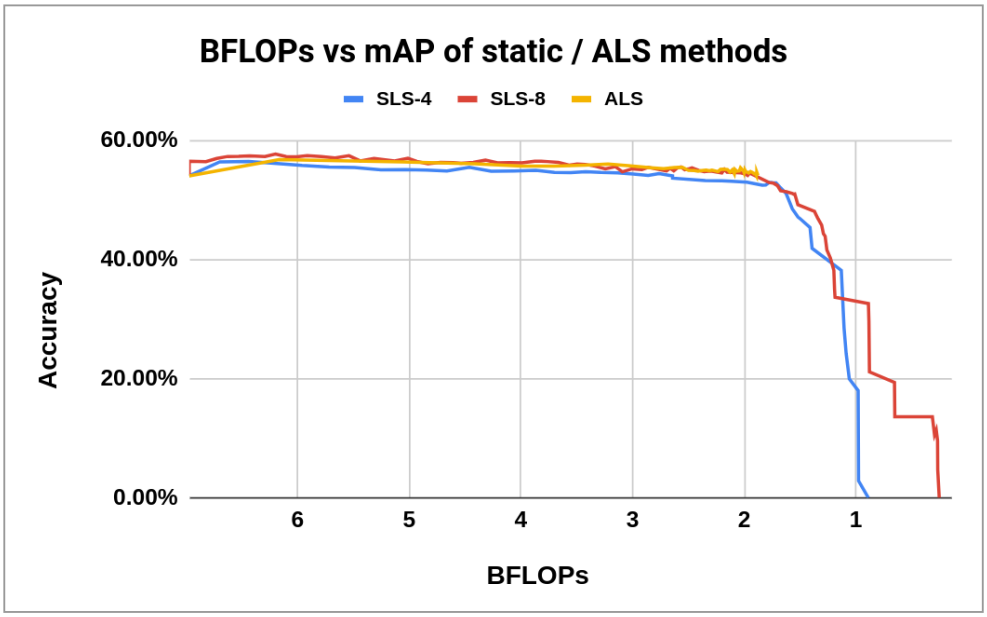

Figure 6.2: Accuracy progression versus computational complexity to compare SLS and ALS pruning method.

6.3 Comparision Adaptive vs Static Layer Sensitivity Pruning

We compared the results of method that statically removed n-blocks and fine-tune at a time, which called static layer sensitivity pruning (SLS). The number of blocks n were set to 4 and 8, and marked as SLS-4 and SLS-8 blocks, respectively. The stop condition of SLS was if all the convolutional layers in the networks left one block number of input channel. We tested using the same dataset and pre-trained weight as Section 6.3.

As shown in Fig. 6.2, SLS-4 sustains the 1% of accuracy drop until BFLOPs of ≤ 1.95. The sustain point of SLS-8 is BFLOPs of ≤ 2.64, which is faster than that of SLS-4. ALS finishes the algorithm without causing an accuracy drop

to the point where the BFLOPs ≤ 1.87. After the accuracy-sustain points, the significant drop occurs as the pruning step progress for the two SLS pruning. This is because, there are only important blocks remained to be pruned. This experiment illustrates two benefits of ALS pruning. First, it can be found automatically just before the sustain point. This means that ALS is able to detect almost all unimportant blocks in the network. The second benefit is to avoid unnecessary fine-tune. In Section 6.2, ALS aggressively removed blocks up to 12 pruning steps, and The computation of complexity is 2.55 BFLOPs. To reduce this amount of computation, SLS-4 should process more than 49 pruning steps. Furthermore, At this point, At this point, the difference in accuracy from ALS is only 0.01%, so the SLS-4 performs fine-tune beyond the need of 30×20 epochs.

Chapter 7

Related Work

Recent software related works on high-performance inference accelerators for CNN can be divided into three categories; matrix decomposition, network quantization and weight pruning. Matrix decomposition [10, 17, 31] reduces parameter dimensions by low-rank approximation using Singular Vector Decomposition (SVD). Network quantization [38, 9, 13] reduces the precision of the weights from single-precision to low-bit such as 8 bits or 1 bit. Weight pruning [36, 33, 32, 34, 21, 6, 16, 35] removes redundant weights connections without critically affecting the accuracy of the original model. In this paper, we focus on weight pruning and plan to integrate other categories in the future.

Recently, researchers have proposed and evaluated [**?**] different granularity of pruning from fine grained sparsity to coarse-grained sparsity. Fine-grained pruning technique [36, 33, 32, 34] prunes individual weights. [32] proposes iterative technique to prune small weights below the threshold after training the model, and further improved compression of the model in [33] adding weights quantization and Huffman coding. [36] extends iterative pruning technique by

restoring previously pruned neurons during fine-tuning. [34] proposes an energy-aware pruning algorithm based on energy consumption cost metric. However, fine-grained pruning does not speedup sparse-matrix libraries such as cuBLAS [1] /cuSPARSE [2] nor enables network compression based on filter deletion.

On the other hand, coarse-grained pruning technique prunes entire input and/or output filters which enables both aspects mentioned above at the cost of a higher impact on the networks accuracy. The key idea of coarse-grained pruning is to evaluate and choose unimportant input/output filters that have little effect on accuracy. There are heuristic techniques to measure the importance of the filters based on weight values [21, 6, 16, 35]. [21] proposes to prune the filters according to the impact of their weights (evaluated with different norms) and to tailor the pruning rate of the each layers based on its information. [6] greedily evaluates the importance of the filters by estimating the network's accuracy after pruning each of them individually in specific layers. [16] measures the value of filter's average percentage of zero (APoZ), then prune the filters with higher values of APoZ. [35] prunes the filters with l_2-norm criteria by putting zeros in the filters every epoch during training phase. Pruned filters have the chance to be updated, since the zero values are updated during back-propagation. There are other techniques to estimate the impact of filters on the model based on training set results [39, 18, 15]. [39] exploits scaling factors in the batch-norm layers [30] that induces channel sparsity. In this experiment, channels having scaling factors under a certain threshold are pruned. [18]'s pruning is based on finding weak channels of a layer and prune them along with the corresponding filters from the previous layer. Weak channels are defined as channels that yield small activation values when processing a set of images. [15] selects the channels to prune based on LASSO regression and uses least square to minimize reconstruction errors.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This paper presented adaptive layer sensitive pruning, an effective pruning technique for the AIX architecture. The AIX is systolic array accelerator for DNN inference, capable of processing different kinds and sizes of neural networks. We pruned block granularity of sparsity, execution unit of AIX, to reduce inference time directly. Adaptive layer sensitive pruning iteratively selects a block in convolutional layer which shows the least impact on the drop of accuracy. Fine-tune is performed one epoch at a time and up to 20 epochs; the parameters with the best accuracy recovery is selected and pruned in following next pruning step. The presented technique prunes to three object detection networks, YOLOv1, YOLOv2, and Tiny-YOLOv2, and achieve state-of-the-art compression ratio within 1% loss of accuracy.

8.2 Future Work

AIX supports 4/8/16 bit quantization mode for each convolutional layer. By exploiting this functionality, the model can be further compressed while maintaining accuracy after pruning. Similar to measuring the layer sensitivity when performing pruning, each layer can be evaluated after quantized independently. On the basis of 8-bit, one that do not affect the accuracy among the layers can be set to 4-bit and vice versa. Instead, if the neural network has n-convolutional layers, there are 3^n combinations cases, so research is needed to reduce the design space search.

There are some limitations in this research. First, block granularity of sparsity is suffered from its coarse-grainedness for neural networks with a small number of input channels in the convolutional layer. For example, the classification neural network, ResNet-20 [14] on CIFAR-10[20] dataset, the convolutional layers with the most input channels only have 64. When AIX quantizes to 8-bits, up to 4 blocks can be grouped to one layer, and if we prune a single block, 25% of the layer information is lost. Second, We statically fix the number of fine-tune epoch. We found that when the fine-tnue reaches a certain level, the accuracy is no longer restored and saturated. That is, saturation may occur before reaching the fixed epoch value, which induce time-inefficient and waste of power. One option to resolve the problem is to make the simple stopping algorithm that detect saturation. In order to propose the algorithm, it takes numerous time to experiment not only object detection but various type of neural networks. However, this will bring not only convenience to users, but also performance improvement.

Deep neural network (DNN) accelerators based on systolic arrays have been shown to achieve a high throughput at a low energy consumption. The regular

architecture of the systolic array, however, makes it difficult to effectively apply network pruning and compression; two important optimization techniques that can significantly reduce the computational complexity and the storage requirements of a network. This work presents AIX, an FPGA-based high-speed accelerator for DNN inference, and explores effective methods for pruning systolic arrays. The techniques consider the execution model of the AIX and prune the individual convolutional layers of a network in fixed sized blocks that not only reduce the weights of the network but also translate directly into a reduction of the execution time of a convolutional neural network (CNN) on the AIX. Applied to representative CNNs such as YOLOv1, YOLOv2 and Tiny-YOLOv2, the presented techniques achieve state-of-the-art compression ratios and are able to reduce interference latency by a factor of two at a minimal loss of accuracy.

Bibliography

[1] cublas cuda toolkit v10.1.243. https://docs.nvidia.com/cuda/cublas/index.html#introduction.

[2] cusparse cuda toolkit v10.1.243. https://docs.nvidia.com/cuda/cusparse/index.html.

[3] Description of xilinx vu13p fpga. https://www.xilinx.com/products/boards-and-kits/1-1177ccc.html.

[4] Geforce rtx 2080 ti. https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2080-ti/.

[5] Intet(r) xeon(r) e5-2620 specification. https://ark.intel.com/content/www/us/en/ark/products/64594/intel-xeon-processor-e5-2620-15m-cache-2-00-ghz-7-20-gt-s-intel-qpi.html.

[6] R. Abbasi-Asl and B. Yu. Structural compression of convolutional neural networks based on greedy filter pruning. *CoRR*, abs/1705.07356, 2017.

[7] M. Ahn, S. J. Hwang, W. Kim, S. Jung, Y. Lee, M. Chung, W. Lim, and Y. Kim. Aix: A high performance and energy efficient inference accelerator

on fpga for a dnn-based commercial speech recognition. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1495–1500, 2019.

[8] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, Jan 2017.

[9] M. Courbariaux, J.-P. David, and Y. Bengio. Training deep neural networks with low precision multiplications. *arXiv e-prints*, Dec. 2014.

[10] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 1269–1277, Cambridge, MA, USA, 2014. MIT Press.

[11] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, Jun 2010.

[12] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang. A survey of FPGA based neural network accelerator. *CoRR*, abs/1712.08934, 2017.

[13] P. Gysel. Ristretto: Hardware-oriented approximation of convolutional neural networks. *CoRR*, abs/1605.06402, 2016.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

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

[16] H. Hu, R. Peng, Y. W. Tai, and C. K. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *CoRR*, abs/1607.03250, 2016.

[17] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *CoRR*, abs/1405.3866, 2014.

[18] L. Jian-Hao, W. Jianxin, and L. Weiyao. Thinet: A filter level pruning method for deep neural network compression. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[19] R. Joseph and F. Ali. Yolo9000: Better, faster, stronger. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[20] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research).

[21] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016.

[22] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[23] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *CoRR*, abs/1810.05270, 2018.

[24] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally. Exploring the regularity of sparse structure in convolutional neural networks. *CoRR*, abs/1705.08922, 2017.

[25] J. Redmon. *Darknet: Open Source Neural Networks in C*, 2013-2020.

[26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[27] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[28] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149, June 2017.

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[30] I. Sergey and S. Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015.

[31] V. Sindhwani, T. N. Sainath, and S. Kumar. Structured transforms for small-footprint deep learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 3088–3096, Cambridge, MA, USA, 2015. MIT Press.

[32] H. Song, P. Jeff, T. John, and W. J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1135–1143, Cambridge, MA, USA, 2015. MIT Press.

[33] H. Song, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv e-prints*, oct 2015.

[34] T.-J. Yan, C. Yu-Hsin, and S. Vivienne. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[35] H. Yang, K. Guoliang, D. Xuanyi, F. Yanwei, and Y. Yi. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pages 2234–2240. AAAI Press, 2018.

[36] G. Yiwen, Y. Anbang, and C. Yurong. Dynamic network surgery for efficient dnns. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 1387–1395, USA, 2016. Curran Associates Inc.

[37] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(10):1943–1955, Oct. 2016.

[38] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016.

[39] L. Zhuang, L. Jianguo, S. Zhiqiang, H. Gao, Y. Shoumeng, and Z. Changshui. Learning efficient convolutional networks through network slimming. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

요약

Systolic 배열에 기반한 심층 신경망 가속기는 적은 에너지 소비와 높은 처리를 가능하게 해준다. 그러나, 일반적인 systolic 배열의 구조는 신경망의 효율적인 압축과 pruning을 어렵게 만든다. 두 최적화 방법들은 신경망의 시간복잡도와 저장 공간을 크게 감소시킨다. 본 논문에는, 심층 신경망 추론을 위한 FPGA 기반 고속 가속기인 AIX를 소개하고, systolic 배열을 위한 효율적인 pruning 방법에 대해서 탐구한다. 이 방법은 AIX의 실행 모델을 고려하며, 신경망의 크기를 줄여 나간다. 또한, 독립적으로 합성곱 신경망 층 내 고정된 크기의 블록을 제거함으로써, AIX 가속기의 합성곱 신경망의 실행시간을 직접적으로 단축시킬 수 있다. YOLOv1, YOLOv2 및 Tiny-YOLOv2와 같은 대표적인 합성곱 신경망에 적용하였고, 제시된 기술은 최신 압축률을 달성하였다. 그 결과, YOLOv2를 최소한의 정확도 손실로 추론 시간을 1.6 배로 줄일 수 있습니다.

주요어: 합성곱 신경망, FPGA 가속기, Systolic 배열, 모델경량화, 아키텍처별 신경망 Pruning

학번: 2018-29331