Hardware-Aware Pruning for FPGA Deep Learning Accelerators

Jef Plochaet, Toon Goedemé
EAVISE-PSI-ESAT, KU Leuven
Jan Pieter De Nayerlaan 5, 2860 Sint-Katelijne-Waver, Belgium
{jef.plochaet, toon.goedeme}@kuleuven.be

Abstract

Pruning has been widely used for deep neural network optimization and compression. In this paper we propose a pruning method to accelerate FPGA implementations of neural networks with the goal of lowering the inference time as much as possible, while taking into account hardware-specific constraints. We use the normalized L2-norm as a measure of filter importance and iteratively prune the network with a predefined pruning stepsize. We extend this method to also prune around residual connections using the maximum normalized L2-norm as a representation of importance for a group of connected channels. We introduce a hardware-aware pruning method for FPGA deep learning accelerators, adaptively pruning the neural network based on the size of the systolic array used to calculate the convolutions. We validate our methods by pruning a model, designed for polyp segmentation in colonoscopy videos, on two different datasets. This results in almost halving the inference time with minimal loss of accuracy on both datasets. We prove that our two contributions yield an extra 30% *increase in processing speed compared to classical L2 pruning.*

1. Introduction

Throughout the years, deep learning models have reached human-level performance on many different applications, including various medical tasks [19]. These models can be used by doctors as an aid or second opinion during medical examinations. In order for people to successfully cooperate with these models, the neural networks must run in real time. However, it is not always easy because, according to the current trend, neural networks are growing in size [24]. Using a powerful graphics processing unit (GPU) could be an option, but this has a financial and environmental downside. Therefore, using embedded devices is more feasible. Embedded devices are less powerful, making it more difficult to achieve real time performance. There exist various techniques for optimizing neural networks and decreasing their inference time, such as quantization and pruning. We focus on the latter.

In this paper we present two pruning strategies where our goal is to lower the inference time of the network on an FPGA as much as possible. We prove the effectiveness of our methods by pruning a neural network designed by Eelbode *et al*. [5] for polyp segmentation in colonoscopy videos. We specifically prune this network for an FPGA implementation using the nearbAI [4] accelerator. The network consists of two parts, a backbone and a recurrent head. Currently, we only look at optimizing the backbone by means of pruning. Eelbode *et al*. use the original Deeplabv3+ [2] architecture with an Xception backbone.

Our proposed method goes as follows. We use the normalized L2-norm to decide what filters are important, the lowest $X\%$ are iteratively pruned. The first novelty we introduce is the ability to prune around residual connections which, due to their complex channel dimension dependencies, are typically seen as an optimization blocker. We use the maximum normalized L2-norm as a representation of importance for a group of connected channels. The corresponding channels get pruned when this maximum L2-norm belongs to the lowest $X\%$. Secondly, we introduce a hardware-aware pruning method, specifically designed for FPGA deep learning accelerators. Our method takes into account how the nearbAI engine processes convolutions and we adjust our pruning according to the size of the hardware systolic array.

2. Related Work

Pruning neural networks is a method to remove parameters, filters, channels or layers that are unimportant to a neural network [8]. Resulting in a smaller network (less parameters) and fewer floating point operations (FLOPs). These effects of pruning make it possible to run large neural networks on embedded devices that have less memory and less computing power.

First, there is an important distinction between unstructured and structured pruning [8]. Unstructured pruning removes singular neurons throughout the entire network. A

high compression rate can be achieved with little to no accuracy loss. The disadvantage of unstructured pruning is that it results in sparse matrices. To actually reduce the inference time, specialized hardware or software is needed. Structured pruning, on the other hand, removes entire channels, filters or layers. This allows for a reduction in inference time without the need of specialized hardware or software.

There exist a lot of different neural network pruning approaches. Vadera and Ameen [23] divide these existing methods into three categories, similarity and clustering methods, sensitivity analysis methods and magnitude based methods.

First, similarity and clustering methods prune weights that are similar to each other. This is based on the idea that weights that are similar, are redundant. Theoretically, the accuracy of the model will be minimally affected by pruning these redundant weights. For example, Ayinde *et al*. [1] cluster the different filters in a convolutional layer where the average similarity is above a certain threshold. From each cluster they select a random filter as a representative filter for that cluster. The representative filter is retained and all other redundant filters in the cluster are pruned.

Second, sensitivity analysis tries to determine the effect the removal of a weight or filter has on the loss of the network. Weights that have the least effect on the loss will be pruned. Molchanov *et al*. [17] uses the first order Taylor expansion to approximate the change in loss after removing a parameter or filter. Parameters are pruned when they have almost no impact on the cost function.

Last, magnitude based methods use the magnitude of the weights to measure their importance. Li *et al*. [13] calculate L1-norm of the weights in a filter, removing the filters with the smallest L1-norm. Molchanov *et al*. [17] uses the L2-norm to determine which filters are important, pruning those with a small L2-norm. He *et al*. [9] argue that using norm-based criteria is only possible when the norm deviation within a layer is large enough and the minimum norm is actually small. They propose using the geometric median. The geometric median contains the common information of all filters in a layer. Pruning filters close to the geometric median has a minimal effect because these filters can be replaced by other filters. The disadvantage of the geometric median approach is that we are unable to compare channels of different convolutions throughout the network as mentioned in [18]. This makes pruning residual connections difficult.

Most pruning algorithms that deal with residual connections, only prune the layers within the residual block and do not prune the residual connections themselves. This is because pruning residual connections is difficult. Residual connections connect different layers across the network. Because of the fact that the dimensions of connected layers must match for the elementwise operation, such connections enforce that all connected layers are pruned simultaneously. Not pruning these residual connections results in an hourglass structure for the residual block, as mentioned by Luo and Wu in [15]. They argue that pruning the residual connections leads to a more accurate model because of the larger pruning space and can result in a faster model even though the number of FLOPs remains the same. Luo and Wu calculate the importance of a channel by removing that channel and using a proxy dataset to evaluate the performance change of the network. For residual connections, they use the same method but remove all connected channels to determine their importance. Gao *et al*. [7] group connected filters and minimize the variance of the weights within the groups applying variance-regularization during training. The weights with a normalized L1-norm smaller than a certain threshold are pruned. Liu *et al*. [14] use Fisher information to determine the importance of a channel. They group different layers that have connected channels. The channels of grouped layers are pruned simultaneously. The overall importance of connected channels follows the chain rule of gradient computation.

Hardware-aware pruning takes into account the specific hardware that will be used or how the network calculations are performed. Shen *et al*. [21] introduce hardware-aware latency pruning (HALP). They compute a look-up table containing the latencies of each layer. First, they rank the neurons according to their importance. Then, they calculate a latency contribution score for each neuron. They group neurons in order of importance and determine which neurons to prune using an augmented knapsack solver. Sun *et al*. [22] discuss a similar approach to ours. They apply hardware-aware pruning on 3D convolutions for FPGAs to match the loop tiling technique. Here, weight tensors are divided into multiple blocks of the size of the FPGA buffer. The network is pruned per weight block. We, on the other hand, select which filters to prune based on the size of the systolic array. Furthermore, we actually remove the filters from the architecture while they use an "enable signal" signal to decide if a block should be skipped. This compresses the size of the network instead of only improving the inference time.

3. Method

The goal is to accelerate the network so that we have a real time FPGA implementation without sustaining a great loss of accuracy.

Our pruning method is an extension of the methods proposed by Ophoff *et al*. [18] and Molchanov *et al*. [17]. Figure 1 shows our pruning pipeline. We start by loading our trained model. Then we calculate the importance of the individual channels throughout the entire network. Next we prune the $X\%$ least important channels and fine-

tune the network. Unlike [18], we select the network that achieves the highest accuracy on the validation set and use this model to start the next pruning iteration. This allows us to go through more pruning iterations and thus further compress the model. However, the risk is that we might slightly overfit on the validation set. Early stopping is implemented during the fine-tuning of the network. When the model reaches a predefined accuracy increase (α), the fine-tuning is stopped and we move on to the next pruning iteration. We repeat this process until the network loses a predefined amount of accuracy (β) on the validation set compared to the original model. The values for α and β are empirically chosen during the experiments. In our case, we prune a segmentation model instead of an object detection model like Ophoff *et al*. [18]. Because the segmentation of objects in images is an unbalanced problem, it is not advised to use pixel accuracy as a metric. Therefore, we use the Dice score.

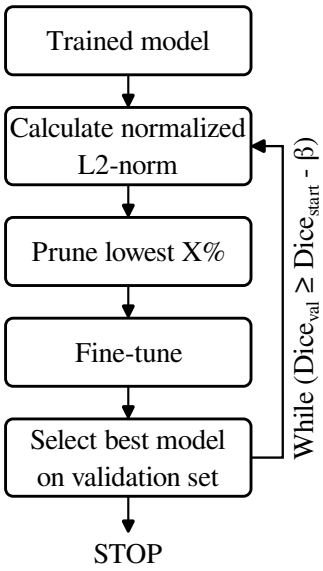

Figure 1. Our pruning pipeline.

To evaluate the importance of the channels in the network, we use the normalized L2-norm (Eq. (1)). Ophoff *et al*. [18] show that this is a simple but effective way to calculate the importance of the channels especially on more constrained datasets. The normalization of the L2-norm allows us to compare channels from different layers across the entire network.

$$Importance(w_i) = \frac{||w_i||_2}{\sqrt{\sum_{w_j \in W} ||w_j||_2^2}} \quad (1)$$

3.1. Pruning Residual Connections

Residual connections are typically seen as optimization blockers. The pruning of layers around residual connections is inherently more difficult because we need to prune the same channels in all the different layers that are involved in the element-wise operation. The connected filters are shown in red in Figure 2. Pruning one of these layers requires pruning the same filters in the other layer, which is typically avoided, such as in [18].

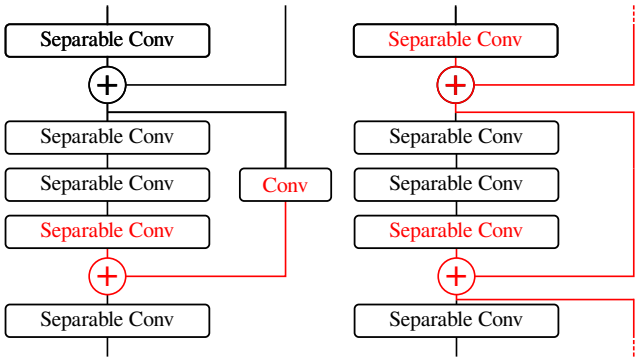

Figure 2. Illustration of residual connections in Deeplabv3+ [2] with (left) and without (right) a convolutional block in the skip connection. Red text shows the connected filters.

However, we propose to build a dependency graph of all layers in the architecture and prune connected groups simultaneously. We compare different channels using the normalized L2-norm. All channels involved in the element-wise operation have their own importance value. This makes it more difficult to decide if or when we need to prune these channels. The decision on when to prune these layers is especially important for the network we are working with. Because this model is based on Deeplabv3+ [2], the structure in Figure 2 (right) is repeated 16 times in the so-called "middle flow". This means that 18 layers are connected together (instead of just 2). Pruning one of these layer means pruning the same filters in all 18 layers. Our decision on when to prune these layers could have a major influence on our pruning results.

According to Luo and Wu [15], pruning these residual connections enlarges the pruning space, which can lead to more compression. Their experiments show that it could also improve inference time even though the multiply accumulates (MACs) are similar to a model where they do not prune the residual layers.

Our proposed solution is to select the maximum L2-norm of the corresponding channels as shown in Equation (2). Here, L contains all the normalized L2-norms of the connected channels and L_{rep} is the maximum value in L.

$$L_{rep} = max(L) \quad (2)$$

We use L_{rep} as a representation of importance for the connected channels so we can compare these channels to the other channels throughout the network. The intuition of

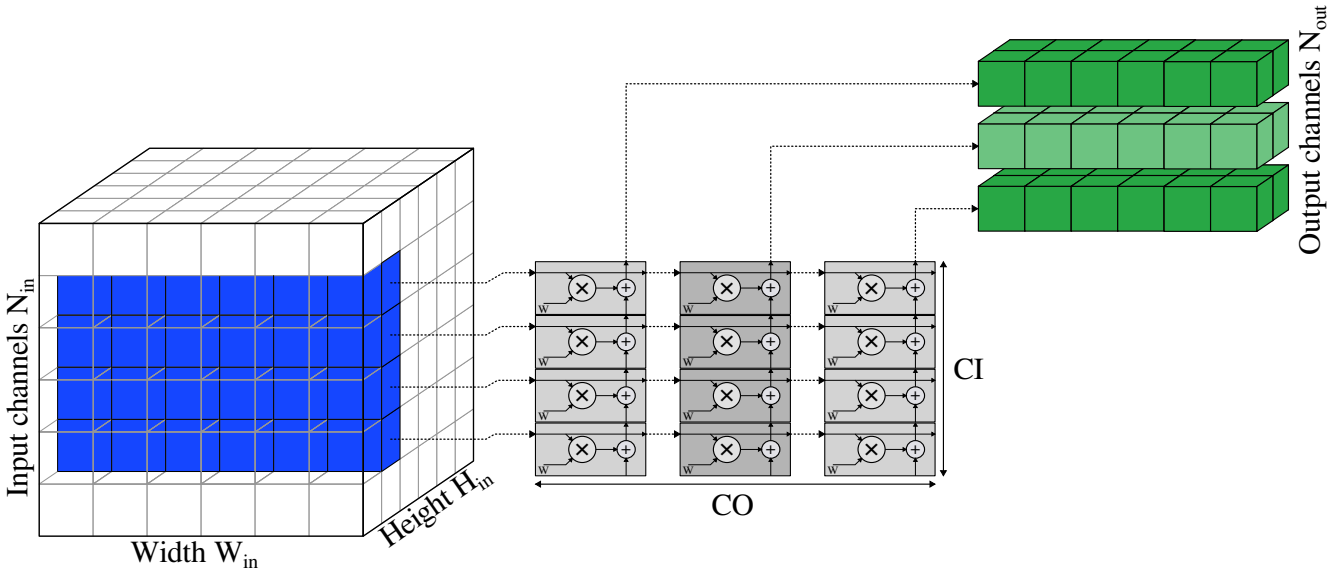

Figure 3. Execution of a 1×1 convolution using a systolic array of MACs.

this approach is that the maximum normalized L2-norm indicates the most important channel. We will only prune the connected channels when this maximum value is in the lowest $X\%$. Otherwise, we may discard channels that contain important information too quickly at the expense of accuracy. If the maximum value is in the lowest $X\%$, all other connected channels are automatically in the lowest $X\%$ as well. This means that these channels are not important and can therefore be pruned with a minimal effect on the accuracy. We also experimented with other alternatives, like using the minimum or average of L as a representation of importance as will be described in Section 4.2.

3.2. Hardware-Aware Pruning (HAP)

In order to optimize our network even further, we propose to make this pruning approach hardware-aware. We use the nearbAI engine [4] from easics [3]. nearbAI is designed to accelerate neural networks for embedded devices such as FPGAs. Our prior knowledge of how convolutions are processed by the nearbAI engine allows us to tailor our pruning specifically for this engine and achieve better pruning results with the aim of achieving the lowest inference time possible. The engine uses a systolic array of multiply accumulates (MACs). Each MAC has two inputs, the first input is connected to a weight. The input activations are shifted from left to right through the systolic array and are delivered to the second input. Every MAC performs the multiplication, adds the result of the multiplication to the result of the element below it and sends that result to the element above.

Figure 3 shows an example of the execution of a 1×1 convolution using a systolic array. Each MAC can execute a 1×1 convolution. Bigger convolutions can be seen as the addition of multiple 1×1 masks. The input of the convolution is a tensor with dimensions $W_{in} \times H_{in} \times N_{in}$, these are the width, height and input channels, respectively. The convolution generates N_{out} output channels. The systolic array has a size $CI \times CO$.

Imagine the systolic array has 1 column ($CO = 1$) and as many rows as there are input channels ($CI = N_{in}$). Pushing one matrix of $W_{in} \times N_{in}$ through the systolic array would result in one row of output activations for one output channel. Now, imagine that the systolic array has as many columns as output channels ($CO = N_{out}$). Pushing the matrix of size $W_{in} \times N_{in}$ through the systolic array results in one row of output activations for every output channel. This process is repeated across the height dimension (H_{in}) to calculate the entire output tensor.

Ideally, a systolic array that has as many rows as input channels ($CI = N_{in}$) and as many columns as output channels ($CO = N_{out}$) is used. This would allow for minimal routing of the weights during the calculation of a convolution. Unfortunately, the size of the systolic array is usually smaller than $N_{in} \times N_{out}$ of a convolution. In practice, nearbAI divides this big "virtual" array of size $N_{in} \times N_{out}$ into multiple smaller arrays of fixed size $CI \times CO$. The different smaller arrays are the different hardware positions the FPGA has to be programmed in to calculate the convolution. The time it takes to perform the convolution depends, among other things, on the number of different positions in which the hardware has to be programmed.

During pruning we remove filters in the convolutional layers which reduces the number of output channels. Since the different hardware positions determine the inference

Pruning stepsize	Residual	HAP	# Parameters	FPGA (ms)	Dice val	Dice test
Original model			41.3M	85.25	0.6465	0.6252
5			14.7M (÷2.81)	57.51 (÷1.48)	0.5970	0.5474
5	✓		13.9M (÷2.97)	53.82 (÷1.58)	0.6058	0.5367
5		✓	12.4M (÷3.33)	52.13 (÷1.64)	0.5944	0.5636
5	✓	✓	10.7M (÷3.86)	49.18 (÷1.73)	0.5978	0.5741
2.5			13.3M (÷3.11)	53.82 (÷1.58)	0.5994	0.5696
2.5	✓		11.1M (÷3.72)	50.16 (÷1.70)	0.5971	0.5684
2.5		✓	10.0M (÷4.13)	49.00 (÷1.74)	0.6050	0.5603
2.5	✓	✓	**8.1M (÷5.10)**	**45.66 (÷1.87)**	**0.6073**	**0.5874**

Table 1. Pruning results on the private dataset.

time of the convolution, we only get a decrease in inference time if we can skip an entire hardware position. To skip entire hardware positions, we propose to prune the filters in multiples of CO.

The pruning pipeline in Fig. 1 remains the same. We calculate the importance of the different filters using the normalized L2-norm. Iteratively, the $X\%$ least important channels, are selected for pruning. Equations (3) and (4) show our hardware-aware pruning implementation. Here, N_{out} is the original number of output channels and N_p is the number of channels of that layer selected for pruning. In Eq. (3), we calculate the number of output channels after pruning ($\hat{N}_{out}$) by finding the nearest multiple of CO closest to $N_{out} - N_p$. The number of output channels after pruning has to be the nearest multiple ($\hat{N}_{out}$).

$$\hat{N}_{out} = \left\lceil \frac{N_{out} - N_p}{CO} \right\rceil \times CO \tag{3}$$

$$\hat{N}_p = N_{out} - \hat{N}_{out} \tag{4}$$

In Equation (4), we calculate the number of channels that will be pruned ($\hat{N}_p$). We select the $\hat{N}_p$ least important channels for pruning. The remainder of the previously selected channels ($N_p - \hat{N}_p$) are retained. Keeping these channels could result in a small accuracy gain that would allow us to prune for more iterations and compress the model even further. Also, immediately removing these channels does not lead to a decrease in inference time because we do not skip an entire hardware position. There is no speed advantage in removing these channels prematurely. If no multiple of CO can be reached, we do not prune that layer in that iteration.

4. Results

We evaluate our methods by pruning the network of Eelbode *et al.* [5] on two different polyp segmentation datasets, a private dataset and the SUN Colonoscopy Video Database [10,16]. The inference times of the different models are simulated on a Intel Arria 10 660 System on Module (SoM) [20] using the nearbAI engine.

4.1. Results Private Dataset

Our private dataset consists out of 131619 frames in total. These frames come from 842 sequences and contain 460 different polyps. The original model trained on this dataset achieved a 0.6252 dice score on the test set. The inference time of this baseline model simulated on the FPGA is equal to 85.25 ms.

We run four different experiments for two different pruning stepsizes, 2.5% and 5% per iteration. First, we prune the original model without pruning the residual connections and without applying our hardware-aware pruning strategy. Second, we also prune the residual connections. Next, we apply our hardware-aware pruning strategy. Finally, we combine the pruning of the residual connections and our hardware-aware pruning (HAP) strategy. During HAP we assume that the nearbAI engine uses a systolic array with $CO = 32$ (see Sec. 3.2). This means that after pruning the resulting output channels will always be a multiple of 32. Every iteration, we retrain the network for up to 10 epochs. The α parameter is set to 0.05. We allow up to 5% accuracy loss on the validation set ($\beta = 0.05$). Pruning neural networks is a trade-off between the accuracy and compression of the model. The decision as to what is more important will depend on the situation. If the accuracy of the model is more important, increase β and vice versa. In our case ($\beta = 0.05$), we prioritize model compression but try to limit the accuracy loss by 5% with respect to the original model.

Table 1 gives an overview of the results of the different experiments. Our best pruning result compresses the number of parameters in the model with a factor of 5.1 and decreases the inference time with a factor of 1.9.

There are several things to notice when viewing these results. First, slower pruning (lower pruning stepsize) leads to greater compression. All the models that were pruned with a pruning stepsize equal to 2.5% have less parameters

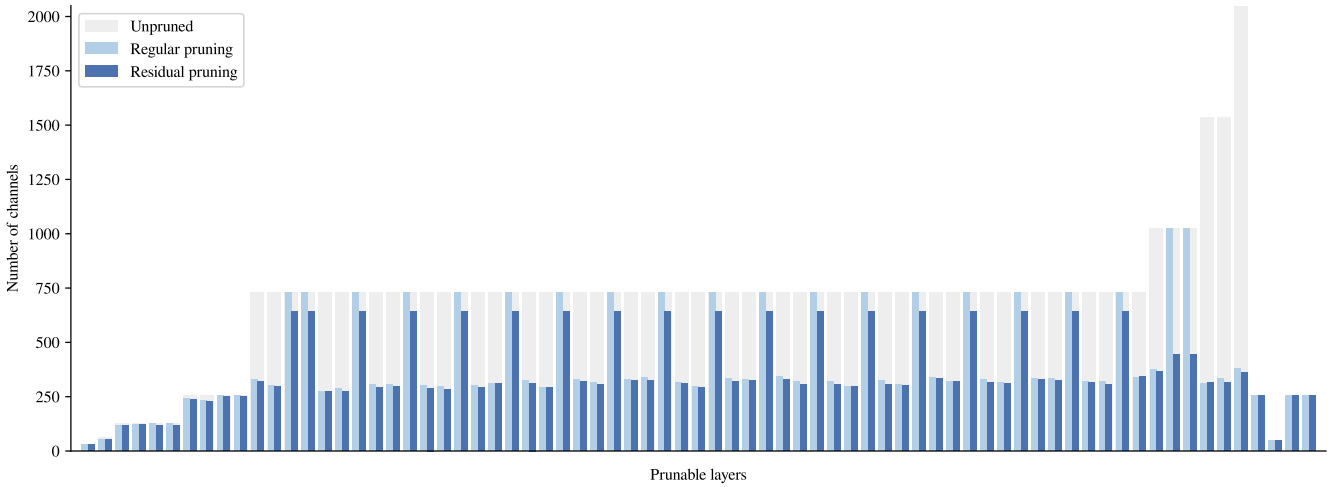

Figure 4. Comparison of pruned channels per layer between the regular pruning and residual pruning with a pruning stepsize of 2.5 for the private dataset.

and a faster inference time compared to their corresponding models that were pruned with a pruning stepsize equal to 5%. By pruning in smaller steps, the model does not lose as much accuracy after pruning. This drop in accuracy can be more easily recovered by retraining compared to when pruning in larger steps. This allows us to prune more iterations and achieve a greater compression.

Secondly, we also observe that pruning the residual connections results in a better compression and faster inference time. This is because pruning the residual connections enlarges the pruning space. In our case, we were able to prune 24 more layers, expanding our pruning space from 49 layers to 73 layers. Figure 4 shows the difference between skipping and pruning the residual connections. Here, the advantage of residual pruning is clear, we are able to simply prune more layers and parameters. It is also remarkable that pruning the residual layers results in a slight decrease in channels in some of the non-residual layers compared to regular pruning. Pruning the layers around the residual connections led to a 10% and 12% speed-up in inference time in comparison to the baseline pruning for pruning stepsizes 5% and 2.5%, respectively.

We also notice that our HAP strategy increases the compression even further. By preserving the channels that do not increase the inference time when removed, we maintain the validation accuracy longer and can prune for more iterations. This can be thought of as pruning in smaller variable steps. This stepsize depends on the the number of multiples of CO that we can prune. Adding the HAP strategy leads to a 16% speed-up in comparison to the pruning baseline for both stepsizes.

Finally, we observe that the combination of both pruning the residual connections and HAP results in the best pruning results. Resulting in a 25% and 29% decrease in inference time compared to classical L2 pruning for pruning stepsizes 5% and 2.5%, respectively. Again, this is due to the larger pruning space and hardware-aware pruning strategy.

The disadvantage of our strategy is that there is a risk to slightly overfit on the validation set as mentioned in Section 3. This hypothesis is confirmed by our results. The average drop in validation accuracy is 0.0460. This is expected because we allow for a 0.05 drop ($\beta = 0.05$). However, the average drop in accuracy on the test set is 0.0618, indicating that we are slightly overfitting on the validation set. This can be avoided by retraining for a set amount of epochs after pruning and selecting the final model instead of selecting the model that performs best on the validation set.

4.2. Results SUN Dataset

We also performed experiments on the SUN colonoscopy video database [10, 16]. This dataset contains 49136 frames taken from 100 different polyps. The original dataset only contains bounding box annotations, but we use the segmentation masks provided by Ge-Peng Ji *et al*. [6, 11, 12]. Training the unpruned network on this dataset resulted in a dice score of 0.5183 and 0.5286 on the SUN-SEG-Easy (Unseen) and SUN-SEG-Hard (unseen) test sets, respectively (see Tab. 2). To enable a fair comparison, the same training parameters were used as in [5] by Eelbode *et al*. Further optimization of the results could be possible, but is beyond the scope of this paper.

In this case, only a pruning stepsize of 2.5% was used. Pruning with a stepsize of 5% caused an excessive drop in validation accuracy that could not be recovered by retraining the network. Also, when pruning with 2.5% we retrain the network for a maximum of 25 epochs to recover the accuracy on the validation set after pruning. Parameters α and

Pruning stepsize	Residual	HAP	# Parameters	FPGA (ms)	Dice val	SUN-SEG-Easy (Unseen) Dice	SUN-SEG-Hard (Unseen) Dice
Original model			41.3M	85.25	0.6633	0.5183	0.5286
2.5			12.9M (÷3.20)	53.21 (÷1.60)	0.6196	**0.5298**	0.5030
2.5	✓		9.9M (÷4.17)	49.09 (÷1.74)	0.6130	0.5265	**0.5324**
2.5		✓	10.2M (÷4.05)	49.18 (÷1.73)	**0.6210**	0.5088	0.5160
2.5	✓	✓	**8.2M (÷5.04)**	**45.71 (÷1.87)**	0.6178	0.5187	0.5323

Table 2. Pruning results on the SUN polyp segmentation database.

β still remain 0.05. We also still assume that the nearbAI engine uses a systolic array where $CO = 32$.

Table 2 contains our results of the pruning experiments on the SUN dataset. Similar to the results on the private dataset, adding residual pruning or HAP produces a better result compared to the baseline pruning. Likewise, the combination of both the pruning of the residual connection and HAP leads to the best compression. We achieve a compression of factor 5.0 and a decrease in inference time of factor 1.9. These results are comparable to the results on the private dataset. But, remarkably, we see a slight increase on the performance on both the test sets. Hence, in this case, the overfitting phenomenon does not occur. The pruning even has a regularization effect. Some examples of the results on the SUN-SEG-Hard (Unseen) test set are shown in Figure 5. We use the pruned model with residual and hardware-aware pruning for generating these segmentation masks.

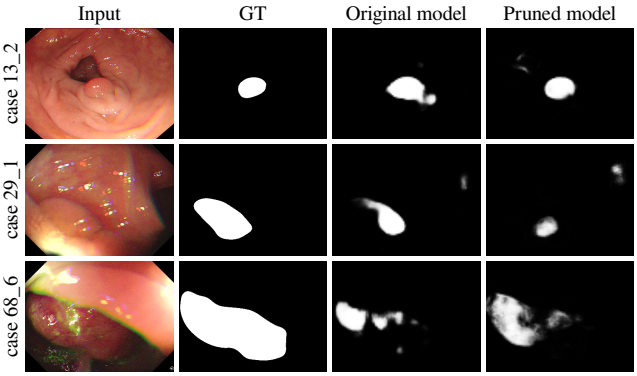

Figure 5. Input, ground truth, result of the original model and result of the pruned model (where we apply residual pruning and HAP) for three examples of the SUN-SEG-Hard (Unseen) test set.

We also performed experiments on different pruning strategies for pruning residual connections. In Section 3.1 we mentioned that we propose to use the maximum L2-norm of the corresponding channels as a representation of importance. We will only prune the connected channels when the maximum L2-norm is in the lowest $X\%$ (2.5% in this case). Here, we compare taking this maximum L2-norm with two variants. First, we use the minimum L2-norm of the connected channels as a representation of importance. Second, we use the average L2-norm of the corresponding channels as a representation of importance. For these experiments, all pruning parameters remain the same and only residual pruning is added. We do not use HAP in this experiment.

	#Parameters	FPGA (ms)
MIN	33.8M	78.13
AVG	13.2M	52.33
MAX	**9.9M**	**49.09**

Table 3. Comparison of using the maximum, minimum and average L2-norm as a representation of importance for pruning of corresponding channels in residual connections.

Table 3 shows the results of this comparison. Using the maximum L2-norm significantly outperforms using the minimum or average. This proves that our intuition that we only prune the connected channels when the most important channel belongs to the least important channels across the model is correct. When pruning channels using the minimum or average, we remove channels that contain important information too fast. As a result, the accuracy on the validation set drops too quickly and less pruning iterations are executed. This reduces the compression of the model and leads to a slower inference time.

5. Conclusion

Our goal was to accelerate deep neural networks for an FPGA implementation by means of pruning. In this paper, we first introduce a way to prune residual connections by using the maximum normalized L2-norm of the connected channels as a representation of importance for the connected channels. The corresponding channels will only be pruned if the maximum L2-norm is in the lowest $X\%$. We also propose a method for hardware-aware pruning for FPGA deep learning accelerators. We adaptively prune the network based on the size of the systolic array used to calculate the convolutions. We demonstrate the effectiveness of our methods by pruning a network dedicated to polyp

segmentation. We train and prune this network on two different datasets, a private dataset and the SUN polyp detection dataset. On both datasets, we almost cut the inference time in half and compressed the number of parameters of the original model by a factor of 5, with only a 5% accuracy drop. Pruning the residual connections and our proposed HAP approach yielded an extra speed-up of almost 30% compared to classical L2 pruning.

One potential opportunity for future work is to compare L2-norm pruning with other pruning methods. Another possibility for future work is to test the effectiveness of our pruning strategies on different datasets.

Acknowledgements

We thank easics for granting us access to their nearbAI tool. Furthermore, we thank Tom Eelbode, who provided the original model and data.

References

[1] Babajide O Ayinde, Tamer Inanc, and Jacek M Zurada. Redundant feature pruning for accelerated inference in deep neural networks. *Neural Networks*, 118:148–158, 2019. 2

[2] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018. 1, 3

[3] easics. easics. https://www.easics.com/. Accessed on: 2022-02-20. 4

[4] easics. nearbai. https://www.easics.com/nearbai/. Accessed on: 2022-02-20. 1, 4

[5] Tom Eelbode, Pieter Sinonquel, Raf Bisschops, and Frederik Maes. Convolutional lstm. *Computer-Aided Analysis of Gastrointestinal Videos*, pages 121–126, 2021. 1, 5, 6

[6] Deng-Ping Fan, Ge-Peng Ji, Tao Zhou, Geng Chen, Huazhu Fu, Jianbing Shen, and Ling Shao. Pranet: Parallel reverse attention network for polyp segmentation. In *International conference on medical image computing and computer-assisted intervention*, pages 263–273. Springer, 2020. 6

[7] Susan Gao, Xin Liu, Lung-Sheng Chien, William Zhang, and Jose M Alvarez. Vacl: Variance-aware cross-layer regularization for pruning deep residual networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. 2

[8] Deepak Ghimire, Dayoung Kil, and Seong-heum Kim. A survey on efficient convolutional neural networks and hardware acceleration. *Electronics*, 11(6):945, 2022. 1

[9] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4340–4349, 2019. 2

[10] Hayato Itoh, Masashi Misawa, Yuichi Mori, Masahiro Oda, Shin-Ei Kudo, and Kensaku Mori. Sun colonoscopy video database. http://amed8k.sundatabase.org/, 2020. Accessed on: 2022-02-20. 5, 6

[11] Ge-Peng Ji, Yu-Cheng Chou, Deng-Ping Fan, Geng Chen, Huazhu Fu, Debesh Jha, and Ling Shao. Progressively normalized self-attention network for video polyp segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 142–152. Springer, 2021. 6

[12] Ge-Peng Ji, Guobao Xiao, Yu-Cheng Chou, Deng-Ping Fan, Kai Zhao, Geng Chen, and Luc Van Gool. Video polyp segmentation: A deep learning perspective. *Machine Intelligence Research*, 19(6):531–549, 2022. 6

[13] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 2

[14] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021. 2

[15] Jian-Hao Luo and Jianxin Wu. Neural network pruning with residual-connections and limited-data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1458–1467, 2020. 2, 3

[16] Masashi Misawa, Shin-ei Kudo, Yuichi Mori, Kinichi Hotta, Kazuo Ohtsuka, Takahisa Matsuda, Shoichi Saito, Toyoki Kudo, Toshiyuki Baba, Fumio Ishida, et al. Development of a computer-aided detection system for colonoscopy and a publicly accessible large colonoscopy video database (with video). *Gastrointestinal endoscopy*, 93(4):960–967, 2021. 5, 6

[17] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016. 2

[18] Tanguy Ophoff, Cédric Gullentops, Kristof Van Beeck, and Toon Goedemé. Investigating the potential of network optimization for a constrained object detection problem. *Journal of Imaging*, 7(4):64, 2021. 2, 3

[19] Sanjay Purushotham, Chuizheng Meng, Zhengping Che, and Yan Liu. Benchmarking deep learning models on large healthcare datasets. *Journal of biomedical informatics*, 83:112–134, 2018. 1

[20] reflex ces. Arria ® 10 soc som. https://www.reflexces.com/modules/intel-arria-10-soc/achilles-som. Accessed on: 2022-02-27. 5

[21] Maying Shen, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose M Alvarez. Halp: hardware-aware latency pruning. *arXiv preprint arXiv:2110.10811*, 2021. 2

[22] Mengshu Sun, Pu Zhao, Mehmet Gungor, Massoud Pedram, Miriam Leeser, and Xue Lin. 3d cnn acceleration on fpga using hardware-aware pruning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020. 2

[23] Sunil Vadera and Salem Ameen. Methods for pruning deep neural networks. *IEEE Access*, 10:63280–63300, 2022. 2

[24] Pablo Villalobos, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, Anson Ho, and Marius Hobbhahn. Machine learning model sizes and the parameter gap. *arXiv preprint arXiv:2207.02852*, 2022. 1