StreamSVD: Low-rank Approximation and Streaming Accelerator Co-design

Zhewen Yu, Christos-Savvas Bouganis
Imperial College London
London, UK
{zhewen.yu18, christos-savvas.bouganis}@imperial.ac.uk

***Abstract*—The post-training compression of a Convolutional Neural Network (CNN) aims to produce Pareto-optimal designs on the accuracy-performance frontier when the access to training data is not possible. Low-rank approximation is one of the methods that is often utilised in such cases. However, existing work considers the low-rank approximation of the network and the optimisation of the hardware accelerator separately, leading to systems with sub-optimal performance. This work focuses on the efficient mapping of a CNN into an FPGA device, and presents StreamSVD, a model-accelerator co-design framework [1]. The framework considers simultaneously the compression of a CNN model through a hardware-aware low-rank approximation scheme, and the optimisation of the hardware accelerator's architecture by taking into account the approximation scheme's compute structure. Our results show that the co-designed StreamSVD outperforms existing work that utilises similar low-rank approximation schemes by providing better accuracy-throughput trade-off. The proposed framework also achieves competitive performance compared with other post-training compression methods, even outperforming them under certain cases.**

I. Introduction

CNNs are widely utilised in image processing and computer vision fields as they outperform their counter-parts and achieve state-of-the-art accuracy in many tasks [1]. In real world, a high-performance image processing system is often required to maximise accuracy with other performance metrics including throughput, latency and energy. As such, growing interest has risen in the deployment of CNNs on specialised hardware and the design of CNN accelerators. Within the accelerator landscape, FPGAs are often targeted as a possible accelerator platform as they can offer advantages in power-sensitive and resource-constrained applications [2], [3].

Redundant CNN models, which are designed for general-purpose GPUs, cannot be efficiently deployed on FPGAs [4]. As such, recent research focuses on the design of tailored light-weight models [5] and the derivation of compressed models to push the accuracy-performance trade-off frontier. In cases where training data are not available due to privacy or security reasons, training-free compression methods, also known as post-training compression, have attracted the attention of the research community.

This work focuses on the topic of post-training compression and accelerator co-design for producing Pareto-optimal design points on the accuracy-throughput frontier. The problem that is addressed here can be formulated as follows: Given a pre-trained CNN model M, the objective is to identify a set of compressed models $\boldsymbol{M'}$ and their corresponding hardware accelerators $\boldsymbol{H'}$ which belong on the Pareto-optimal accuracy-throughput trade-off (A, T) for a target FPGA device D without the possibility of a model retraining step.

Popular post-training compression methods mainly include pruning, quantisation and low-rank approximation [4]. Pruning compresses a pre-trained model by removing unimportant connections between neurons, where quantisation reduces the wordlength of the variables that store the weights and activations in the model. Low-rank approximation decomposes the weight matrices in the model through matrix factorisation and replaces decomposed matrices with their low-rank versions, reducing the number of operations and memory storage. This work focuses on low-rank approximation, more specifically on Singular Value Decomposition (SVD), for addressing the post-training compression problem.

So far, existing work that utilises SVD low-rank approximation develops the compression algorithms and the hardware accelerators separately [6], [7]. As the existing compression algorithms are driven solely by reducing the number of operations and the number of parameters in the model, their solutions lead to sub-optimal designs [8]. Furthermore, a number of them aim to design a general-purpose accelerator instead of customising the hardware for the compute structure and memory requirement of the low-rank approximated model. To address these two issues, we propose StreamSVD, a framework that considers the SVD low-rank approximation algorithm and the accelerator's architecture simultaneously with a focus on a streaming accelerator architecture suitable for throughput maximisation. Our main contributions are:

- A novel fine-grained low-rank approximation algorithm that is tailored for a streaming architecture accelerator, producing design points on the accuracy-throughput frontier for a given set of FPGA resources. Key to our approach is that utilises accelerator information including the latency and folding factor to guide the decisions on the low-rank approximation, leading to a hardware-aware compression.
- We demonstrate that low-rank approximation and quantisation approaches can be efficiently combined through an iterative methodology. Our proposed approach exposes the quantisation error in the decomposition stage itera-

[1] https://github.com/Yu-Zhewen/StreamSVD

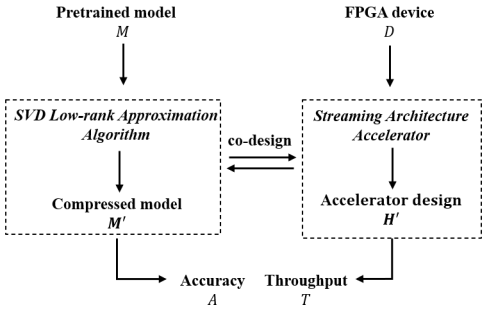


Fig. 1: The overview of proposed StreamSVD framework

tively, reducing the overall impact of quantisation.

- We analyse the advantages on using a streaming architecture for mapping the low-rank approximated CNN as opposed to other architectures, and we demonstrate that the streaming architecture introduces less overhead.

II. Related Work

A. Pruning

Pruning aims to remove unimportant connections between neurons. In order to evaluate the importance of a connection, a common method is to compare the magnitude of the weights corresponding to the connection [9], [10]. Alternatively, Molchanov *et al.* has proposed a Taylor-expansion importance criterion which compares the gradient of the loss function with respect to weights [11]. The importance evaluation and the pruning can be carried out on each connection individually, which creates an unstructured sparsity. As such, the acceleration of such sparse CNN models requires customising the data transfers [12]. To reduce the effort of hardware customisation, pruning approaches have been proposed that are more coarse-grained and structured [13], [14]. For example, channel pruning has been widely supported for various accelerators [15]. We refer readers to [4] for further details on pruning algorithms and the impact on hardware accelerator designs.

In this work, we analyse the similarity between channel pruning and low-rank approximation, and demonstrate how the Taylor-expansion channel pruning algorithm can be adapted for low-rank approximation.

B. Quantisation

Early work on fixed-point quantisation utilised the same wordlength and the same scaling factor across the whole network [16], [17], leading to simple and efficient hardware implementations. As the dynamic range of weights and activations varies between different parts of the network, per-layer and per-channel quantisation have been proposed, also known as Block Floating Point (BFP) quantisation approaches, which allow each layer and each channel to have a corresponding scaling factor [18], [19]. Wang *et al.* [20] applied mixed precision quantisation to solve the dynamic range problem, where the mixed precision quantisation has been implemented through bit-serial computations [21]. We refer readers to [4], [22] for the state-of-the-art quantisation methods and their hardware designs.

Previous work claimed the quantisation and SVD low-rank approximation are orthogonal without providing further insight on how they can be combined [23]–[27]. We revisit this claim and an iterative combination of quantisation and SVD is proposed that utilises SVD to compensate the quantisation error in a retraining-free way.

C. SVD Low-rank Approximation

Qiu *et al.* [18] utilised SVD to decompose the 2-d weight matrix of a fully connected layer. For the convolutional layer, the 4-d convolution weight matrix is unfolded into 2-d so that SVD can be applied to the unfolded matrix [28], [29]. Alternatively, Jaderberg *et al.* [30] and Wang *et al.* [31] sliced the 4-d convolution weight matrix to obtain a group of 3-d matrices. Afterwards, each 3-d matrix was independently unfolded to 2-d and then decomposed by SVD. In this paper, these different methods of applying SVD to the convolutional layer are referred to as "decomposition schemes".

Although previous work has explored various decomposition schemes, they do not explore the possibility of tailoring each convolutional layer to a specific scheme, but rather impose all layers to adopt the same scheme. By contrast, this work allows each convolutional layer to be mapped to the most appropriate decomposition scheme.

In terms of selecting the decomposition rank across the layers in a model, Tai *et al.* proposed a trial-and-error method [29]. They selected the smallest decomposition ranks for which accuracy degradation was smaller than 1%. To address the long execution time of the trial-and-error approach, Zhang *et al.* [32] proposed a one-shot method which compares the magnitude of eigenvalues. This eigenvalue method simplified the rank selection problem and has been widely used in recent research [7], [33]. This work proposes a novel rank selection method based on Taylor-expansion, which outperforms the eigenvalue method.

For hardware implementations, existing work has deployed low-rank approximated CNNs on accelerators with the single computation engine architecture [6], [7]. Single computation engine only has one set of processing elements (PEs) which is time-shared among all the layers. Weights and activations are usually stored on the off-chip memory [2]. As a result, existing work is more likely to be memory-bound because low-rank approximation increases the depth of a CNN. In addition, single computation engine cannot tailor the hardware to the computation of every layer, which restricts the choice of decomposition schemes. This work considers the low-rank approximation algorithm and the streaming architecture jointly to solve these problems. Streaming architecture allocates dedicated PEs to each layer in order to construct a layer-wise pipeline [2]. Most streaming architecture designs store weights on-chip and buffer activations with FIFOs to reduce off-chip memory accesses [34].

TABLE I: Decomposition schemes considered in StreamSVD. "->" indicates two low-rank layers are sequentially connected.

scheme (S)	original weight matrix	unfolded matrix	two low-rank layers
s_0 [35]	$M^{F\times C\times K\times K}$	F groups of $W^{KK\times C}$	$Conv(1\times 1, C, RF, 1)$ -> $Conv(K\times K, RF, F, F)$
s_1 [28]		$W^{F\times CKK}$	$Conv(K\times K, C, R, 1)$ -> $Conv(1\times 1, R, F, 1)$
s_2 [29]		$W^{FK\times CK}$	$Conv(1\times K, C, R, 1)$ -> $Conv(K\times 1, R, F, 1)$
s_3 [31]		C groups of $W^{F\times KK}$	$Conv(K\times K, C, CR, C)$ -> $Conv(1\times 1, CR, F, 1)$

III. SVD Low-rank Approximation Algorithm

A. SVD on Convolution

Consider a convolutional layer $Conv(K\times K, C, F, G)$, where K, C, F, G denote the spatial kernel size, the number of input feature maps, the number of output feature maps and the number of groups. Its 4-d weight matrix $M^{F\times C\times K\times K}$ is firstly unfolded into the 2-d matrix W, following the decomposition scheme S. Afterwards, W is decomposed into two full-rank matrices W_1, W_2 by SVD. The process of decomposition is equivalent to splitting $Conv(K\times K, C, F, G)$ into two sequential-connected convolutional layers, which are referred to as full-rank layers in the rest of this paper.

$$W = U\Sigma V^* = (U\Sigma^{\frac{1}{2}})(\Sigma^{\frac{1}{2}}V^*) = W_2W_1 \tag{1}$$

After the decomposition, only the R largest singular values are kept in Σ while the rest are truncated. Therefore, W_1, W_2 are approximated by rank-R matrices $\hat{W}_1$, $\hat{W}_2$, which is equivalent to replacing two full-rank layers with two low-rank layers.

To obtain optimal compression results, the key is to determine the decomposition scheme S and the decomposition rank R for every convolutional layer in the given model M.

B. Select Decomposition Scheme

Table I summarises four different decomposition schemes considered in StreamSVD. For every convolutional layer, our framework is able to select the most appropriate scheme from the table. As (2), StreamSVD defines the most appropriate scheme as the one which introduces the smallest l_2 error under the same number of operations.

$$\min_{S'\in\{s_0,s_1,s_2,s_3\}} ||W - \hat{W}_2\hat{W}_1||_2$$
$$s.t.\ OPs(\hat{W}_1) + OPs(\hat{W}_2) \le OPs_{avail} \tag{2}$$

$OPs(\hat{W}_1)$ and $OPs(\hat{W}_2)$ denote the number of operations in two low-rank layers, while OPs_{avail} specifies the operations budget allocated to these two low-rank layers. StreamSVD uses a majority vote to decide the best scheme after a sweep of OPs_{avail}, as Algorithm 1, where $OPs(M^{F\times C\times K\times K})$ represents the number of operations in the original convolutional layer before low-rank approximation.

Our scheme selection algorithm is based on the following assumptions:

- The per-layer choice of the best scheme is independent with each other. In addition, the optimal scheme is irrelevant to the value of OPs_{avail}.

Algorithm 1 Scheme selection

Input: the original weight matrix $M^{F\times C\times K\times K}$
Output: selected scheme S
1: **for** $OPs_{avail} \in (0, OPs(M^{F\times C\times K\times K}))$ **do**
2: calculate (2) to determine S'
3: **end for**
4: **return** $S = majority(S')$

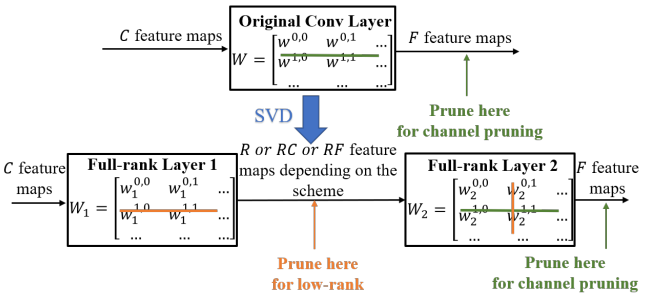


Fig. 2: The relationship between channel pruning and low-rank approximation

- The l_2 distance between the unfolded weight matrix W and the product of low-rank weight matrices $\hat{W}_1$, $\hat{W}_2$ can indicate the level of accuracy degradation caused by low-rank approximation.

C. Select Rank

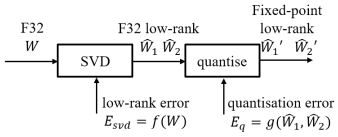
After determining the decomposition scheme, the next step is selecting the decomposition rank. Our method is interpreting the rank selection problem as a special form of pruning. As Fig. 2 shows, reducing ranks of W_1, W_2 is equivalent to removing feature maps between two full-rank layers [36].

Our method firstly evaluates the importance of a feature map f between two full-rank layers by the Taylor pruning criterion [11], which is

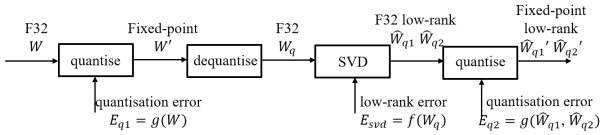
$$I_f = \sum_{w\in f} (w\frac{\partial L}{\partial w})^2, \tag{3}$$

In (3), w is the element of W_1. L is the loss function of the decomposed CNN, whose value is computed on the validation dataset.

StreamSVD then obtains the importance of the rank by summing the corresponding feature maps' importance, which is shown in (4). Specifically, in s_1 and s_2, each rank corresponds to one feature map, while for s_0 and s_3, each rank corresponds F and C feature maps respectively because these two decomposition schemes contain group convolutions.



(a) quantisation after SVD



(b) quantisation before SVD

Fig. 3: Exchange the order of quantiation and SVD

$$I_r = \sum_{f \in r} I_f \tag{4}$$

After determining the per-rank importance, the rest of our rank selection algorithm can be described as follows:

- For each pair of low-rank layers, their rank is initialised as R_{lim} which ensures the number of operations will not increase after low-rank approximation, i.e., $OPs(\hat{W}_1) + OPs(\hat{W}_2) \leq OPs(M^{F \times C \times K \times K})$.
- The per-rank importance I_r is sorted globally in the whole network.
- Keep removing the rank with the smallest importance until the targeted compression ratio is met, which reduces R_{lim} to the selected rank R.

D. Quantisation

Apart from SVD low-rank approximation, StreamSVD also compresses the network through quantisation. This section investigates how to combine low-rank approximation with quantisation. Through a qualitative analysis, we demonstrate quantisation and low-rank approximation are not orthogonal in terms of the errors that they introduce. Therefore, StreamSVD adopts an iterative approach to efficiently combine these two compression methods.

1) Sequential Compression: Fig. 3 compares two flows which sequentially compress a CNN with both of quantisation and low-rank approximation.

In flow (a), W is firstly approximated by low-rank matrices $\hat{W}_1$, $\hat{W}_2$, which injects the low-rank approximation error E_{svd} into the convolution operation. This error is related to singular values and singular vectors of W. Therefore, it can be represented as a function of the unfolded weight matrix $f(W)$. Afterwards, low-rank matrices $\hat{W}_1$ and $\hat{W}_2$ are quantised into $\hat{W}_1{}'$ and $\hat{W}_2{}'$. The quantisation error is related to the dynamic range of parameters in $\hat{W}_1$, $\hat{W}_2$. Overall, the superposition of the low-rank approximation error and the quantisation error is

$$E_a = f(W) + g(\hat{W}_1, \hat{W}_2) \tag{5}$$

In contrast, flow (b) swaps the order of low-rank approximation and quantisation. W is firstly quantised to W' before being low-rank approximated. However, this is not the only place that involves quantisation. The decomposition of fixed-point numbers introduces a second quantisation error. In addition, the low-rank approximation error E_{svd} in flow (b) is a function of the quantised weight W_q rather than the original weight W. The overall compression error of flow (b) can be represented as

$$E_b = g(W) + f(W_q) + g(\hat{W_{q1}}, \hat{W_{q2}}) \tag{6}$$

Our experiment in section VI-B reveals E_a is evidently smaller than E_b. Quantisation and low-rank approximation are not completely orthogonal in terms of the error they introduce. Therefore, the sequential combination of these two compression methods is sub-optimal.

2) SVD Iterative Quantisation: Motivated by the previous analysis, StreamSVD iteratively combines the SVD low-rank approximation and quantisation, which is demonstrated in Fig. 4. This iterative combination is inspired by the refinement mechanism of SVD [37], [38].

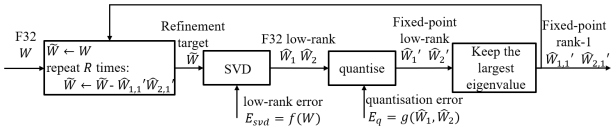


Fig. 4: The proposed iterative combination of quantisation and SVD

As Fig. 4, our method converts the process of SVD into a refinement loop containing R iterations. Each iteration aims to approximate the "refinement target" $\tilde{W}$ by producing two quantised rank-1 matrices. In the first iteration, $\tilde{W}$ is initialised as the unfolded convolution weight W. Afterwards, the residual between the current "refinement target" and the product of two quantised rank-1 matrices becomes the "refinement target" in the next iteration. At the end of the loop, all quantised rank-1 matrices are concatenated to constitute two quantised rank-R matrices which are used as the weights of two low-rank layers.

IV. Streaming Architecture Accelerator

A. Extension of fpgaConvNet

StreamSVD utilises fpgaConvNet [39] as the back-end tool to deploy the compressed CNN onto the FPGA. fpgaConvNet takes the compressed CNN as the input and constructs a layer-wise pipeline, where the computation is performed in a streaming manner. In order to maximise the pipeline's throughput under the resource constraints, fpgaConvNet applies a set of transformations to the network.

The transformation defines the level of parallelism and resource re-use. One important transformation technique in fpgaConvNet is coarse-grained folding, which time-shares PEs between feature maps of a layer. Each PE contains the local memory for storing weights and the dedicated multiply-accumulate unit for computing the corresponding feature map [40]. Coarse-grained folding can be applied to both input feature maps and output feature maps of a convolutional layer.

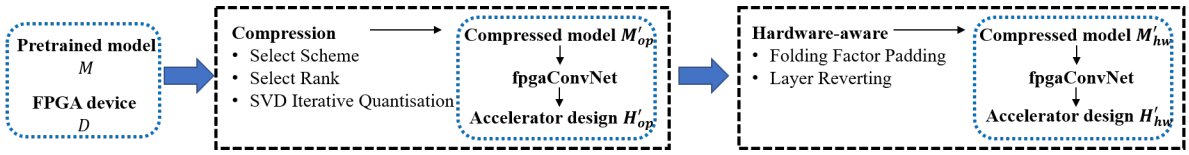


Fig. 5: Complete flow of StreamSVD

The level of the folding is defined by the transformation parameter "coarse-grained folding factor", which is the ratio between the number of PEs and the total number of feature maps F to compute.

$$f_{coarse} = \frac{N^{PE}}{F} \tag{7}$$

StreamSVD extends the idea of coarse-grained folding to support the parallelism inside the group convolution, which is required by s_0 and s_3 in Table I. Take s_3 as an example, the decomposition scheme contains a low-rank layer which is $Conv(K \times K, C, CR, C)$. This layer has C groups where each group contains one input feature map and R output feature map. StreamSVD is able to not only fold between different feature maps but also fold between different groups. By folding the groups, StreamSVD efficiently accelerates the group convolution using the streaming architecture.

Apart from introducing a new dimension of folding, StreamSVD also extends fpgaConvNet to provide accelerator performance feedback, which makes the SVD low-rank approximation hardware-aware. StreamSVD introduces two hardware-aware techniques: folding factor padding and layer reverting.

B. Folding Factor Padding

The inverse of coarse-grained folding factor is proportional to the latency of computing all F feature maps. If this inverse is integer and there is no pipeline stall, all PEs allocated to the layer will be fully utilised in every clock cycle. However, since the number of feature maps in the low-rank layer is related to the rank R, the inverse of coarse-grained folding factor will not be integer unless R is divisible by N^{PE}.

As such, StreamSVD deploys the compressed CNN onto the FPGA and gathers the folding factors of low-rank layers. Using these folding factors, StreamSVD adjusts the SVD low-rank approximation by padding the rank R to R_{pad},

$$R_{pad} = \left\lceil \frac{R}{lcm(R \cdot f_{coarse}^{1,out,R},\ R \cdot f_{coarse}^{2,in,R})} \right\rceil \tag{8}$$

"lcm" stands for Least Common Multiple. For each pair of low-rank layers, $f_{coarse}^{1,out,R}$ is the folding factor for output feature maps of the first low-rank layer and $f_{coarse}^{2,in,R}$ is the folding factor for input feature maps of the second low-rank layers.

The purpose of folding factor padding is using the accelerator feedback to constrain the choice of the rank, which further encourages the PE utilisation on hardware. From another perspective, the padding reduces the compression error while maintains the throughput of the design. Compared with heuristic padding method such as simply padding R to the nearest power of two, our padding method is more device-dependent and model-dependent because of folding factors.

C. Layer Reverting

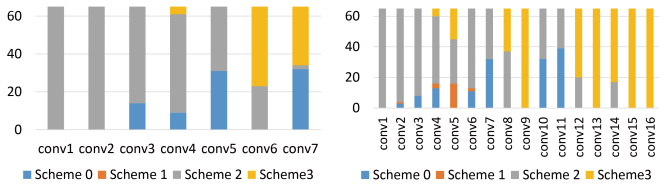
StreamSVD also collects the latency feedback from the accelerator. When the latency sum of two low-rank layers is larger than the latency of the original convolutional layer, it implies SVD does not bring throughput speed-up on hardware. In such cases, the low-rank approximation on that layer will be reverted.

Although low-rank approximation reduces the number of operations and the number of parameters in the CNN, it also introduces the overhead by making the CNN deeper. SVD splits each convolutional layer of M into two low-rank layers. The streaming architecture avoids extra off-chip accesses between two low-rank layers. However, the overhead remains in the extra accumulation units and extra buffers for handling the intermediate data between two low-rank layers. In addition, the reduction in weight parameters does not necessarily lead to the reduction in the BRAM utilisation on FPGA. Therefore, layer reverting helps StreamSVD to identify the layers which do not benefit from SVD.

V. System Workflow

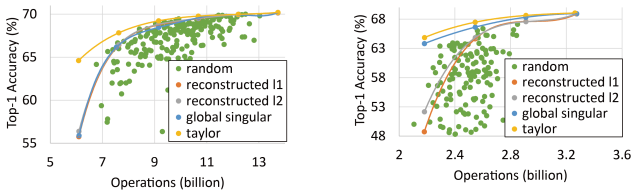
After introducing our compression algorithm and accelerator design, the complete workflow of StreamSVD can be described as follows:

- The given CNN model M is firstly compressed by our low-rank approximation algorithm, which involves selecting the proper decomposition scheme and decomposition rank for every convolutional layer. Afterwards, the quantised weight parameters are generated through our iterative approach. As such, a set of compressed models $\boldsymbol{M'_{op}}$ with different compression ratios are generated.
- For each compressed model, it is deployed on the target FPGA D by fpgaConvNet. This deployment outputs the accelerator design $\boldsymbol{H'_{op}}$ alongside the performance information including the latency and folding factors.
- The accelerator performance information is used to adjust $\boldsymbol{M'_{op}}$ according to our hardware compression techniques including folding factor padding and layer reverting. This adjustment converts $\boldsymbol{M'_{op}}$ to hardware-aware compressed models $\boldsymbol{M'_{hw}}$.
- $\boldsymbol{M'_{hw}}$ are mapped to the FPGA, which generates the final accelerator designs $\boldsymbol{H'_{hw}}$.



(a) VGG11-BN (b) ResNet-18

Fig. 6: Majority voting results of the scheme selection



(a) VGG11-BN (b) ResNet-18

Fig. 7: Selecting ranks with different criteria

The objective of our framework is $\boldsymbol{M'_{hw}}$ and $\boldsymbol{H'_{hw}}$ can optimise the accuracy-throughput trade-off.

$$\underset{\boldsymbol{M'_{hw}},\boldsymbol{H'_{hw}}}{\arg\max}\ [A(\boldsymbol{M'_{hw}}), T(\boldsymbol{M'_{hw}}, \boldsymbol{H'_{hw}})] \tag{9}$$

VI. Evaluation

The performance of StreamSVD is evaluated on pre-trained VGG11-BN, VGG16 and ResNet-18 which come from PyTorch. The accuracy numbers are gathered on the ImageNet validation set. The throughput numbers are generated by fpgaConveNet targeting a Xilinx ZC706 board. [2]

A. Select Schemes and Ranks

We firstly demonstrate the proposed scheme selection and rank selection methods are superior to the existing SVD algorithm.

According to Algorithm 1, we iterated over 65 different values of OPs_{avail} and the results of the majority voting are demonstrated in Fig. 6. Our method is able to directly determine the best decomposition scheme for most convolutional layers. However, for those layers (conv5, conv6, conv7 in VGG11-BN and conv7, conv8, conv10, conv11 in ResNet-18) which do not have an evident winner from the majority voting, top-2 schemes are implemented on hardware and the one that shows the best performance is finally picked.

Fig. 9 demonstrates the performance gain of our per-layer scheme selection compared with "uniform scheme". The "uniform scheme" forces all the layers in the network to adopt the same scheme. As this paper considers four different schemes, we obtained four different trade-off curves of "uniform scheme" and only demonstrate the best one in Fig. 9. Our results imply there is no such a decomposition scheme that is globally optimal for all the layers in a CNN. Therefore, it is beneficial to select schemes in a fine-grained and per-layer way.

In terms of the rank selection, our Taylor-expansion method is compared against four other methods:

global singular: Proposed by Zhang *et al.* [32], the importance of rank R is estimated by $\frac{\Delta\varepsilon/\varepsilon}{\Delta C}$. $\Delta\varepsilon$ is the R-th largest eigenvalue of W^TW and ε is the sum of all eigenvalues of W^TW. ΔC represents the OPs reduction after decreasing the rank by one.

reconstructed $l1$: Estimate the importance of rank R by $l1$ reconstruction error $||W - \hat{W}_2\hat{W}_1||_1$

reconstructed $l2$: Estimate the importance of rank R by $l2$ reconstruction error $||W - \hat{W}_2\hat{W}_1||_2$

random search: Randomly choose the rank R in each layer. Theoretically, this method can find the optimal rank as long as enough searching time is given. However, it is usually impractical to use this method as the searching time grows exponentially with the depth of the network. Fig. 7 contains 200 data points for this method.

From Fig. 7, the proposed rank selection method is significantly better than others and its advantage is more obvious when the compression ratio is high.

TABLE II: Comparison with Quantisation Only. (BitOPs = OPs $\times$ weight wordlength $\times$ activation wordlength)

model	method	parameter size ($\times 10^9$ bit)	BitOPs ($\times 10^{12}$)	Top-1 Accuracy (%)
VGG11-BN	f32	4.251	-	70.41
	w8a16	1.063	1.954	67.69
	w6a16	0.797	1.465	0.12
	StreamSVD (w8a16)	1.048	1.465	67.60
ResNet-18	f32	0.374	-	69.64
	w8a16	0.094	0.466	69.05
	w6a16	0.070	0.349	53.41
	StreamSVD (w8a16)	0.066	0.349	67.51

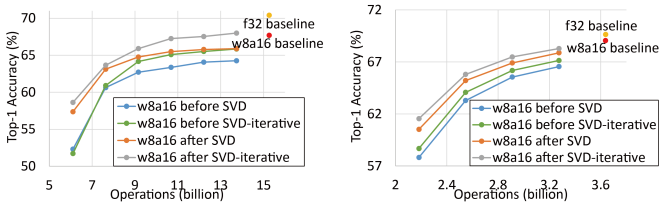
B. Quantisation

This section evaluates the iterative combination of quantisation and low-rank approximation in StreamSVD. The quantisation in our framework utilises the fixed-point representation which is linear asymmetric and without clipping.

We firstly demonstrate our method achieves higher accuracy than using quantisaion only under the same BitOPs. From Table II, both VGG11-BN and ResNet-18 suffer significant accuracy degradation after quantising weights to 6 bits. However, with the same BitOPs, StreamSVD achieves 67.60% and 67.51% accuracy on VGG11-BN and ResNet-18 respectively.

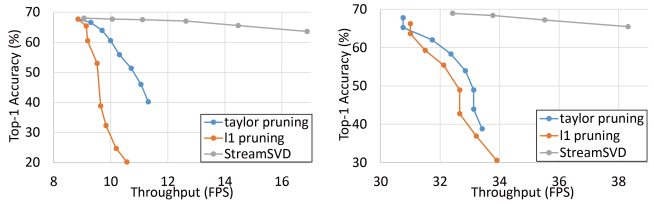
In addition, comparing "w8a16 after SVD" and "w8a16 before SVD" which correspond to flow (a) and flow (b) in Fig. 4, we find the order of applying quantisation and low-rank approximation matters. A model which is firstly low-rank

[2]fpgaConveNet is responsible for mapping each compressed model to hardware. During this mapping, fpgaConvnet searches the accelerator design space to find the throughput-optimal solution. For VGG11-BN and ResNet-18, the searching takes about 0.3 and 0.5 hour respectively using the single core of Intel i7-9700 @3.00GHz. After the searching, fpgaConvnet invokes the HLS tools to generate actual hardware, which can take up to days.



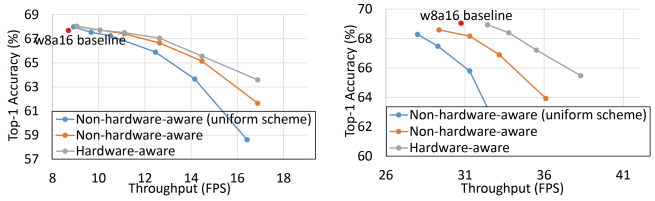
(a) VGG11-BN (b) ResNet-18

Fig. 8: Combing SVD and quantisation. "w8a16" stands for 8-bit weights and 16-bit activations



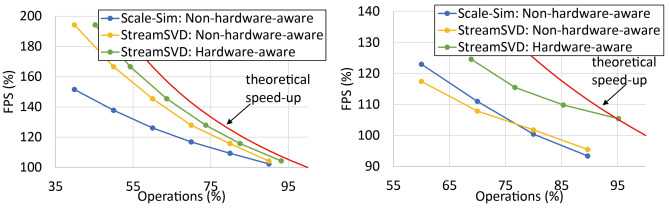
(a) VGG11-BN (b) ResNet-18

Fig. 10: Channel pruning without fine-tuning. Each data point on the pruning curve removes 64 filters from the network.



(a) VGG11-BN (b) ResNet-18

Fig. 9: Accuracy-Throughput trade-off achieved by M'_{op} and M'_{hw}. Top-right points are preferred. "uniform scheme" forces all the layers to have the same decomposition scheme



(a) VGG11-BN (b) ResNet-18

Fig. 11: The contribution of OPs reduction to throughput speed-up

approximated and then quantised achieves higher accuracy than the other way around.

Finally, we confirm our iterative approach, which is annotated as "w8a16 after SVD-Iterative" in Fig. 8, is superior to other attempts of combining quantisation and low-rank approximation. Our iterative approach utilises SVD to regenerate weights and compensate the quantisation error through the residual operation in the refinement loop.

We notice the performance gain of our iterative approach decreases, as the compression ratio of the network increases. When the compression ratio becomes higher, the rank of SVD is smaller, which leads to fewer iterations in the refinement loop. We also observe our iterative approach is more beneficial on VGG11-BN because VGG11-BN has a larger quantisation error than ResNet-18.

C. Hardware-aware Compression

In this section, we evaluate the effect of hardware-aware compression by comparing the performance of M'_{op} and M'_{hw}.

From Fig. 9, although M'_{op} always have smaller OPs than the baseline, some of compressed models actually become slower after the low-rank approximation. As M'_{op} have no knowledge on hardware, it is possible that the overhead overwhelms the benefit of low-rank approximation. By contrast, with the help of hardware-aware techniques, M'_{hw} always have higher throughput than the baseline and they achieve better performance trade-off than M'_{op}.

D. Channel Pruning versus Low-rank Approximation

We also compare our method with channel pruning in Fig. 10. Without retraining, channel pruning causes evident accuracy degradation and brings limited speed-up on fpga-ConvNet. Instead, StreamSVD provides up to 1.94x speed-up with 4.10pp accuracy loss on VGG11-BN and 1.25x speed-up with 3.57pp accuracy loss on ResNet-18.

As discussed before, low-rank approximation can be viewed as a special type of channel pruning on the decomposed network. Our results show pruning the decomposed network is better than directly pruning the original network, when training data are not available. This finding indicates the decomposition makes the network potentially easier to be compressed.

E. Comparison with Other Frameworks

Finally, we compare our framework with other FPGA accelerators in Table III. As the target device and resource utilisation vary in different work, we define an efficiency metric which normalises the throughput by the number of DSPs and the clock frequency. Our framework outperforms existing designs that exploit low-rank approximation by achieving higher efficiency and better accuracy-throughput trade-off. Furthermore, our method is competitive with other compression methods such as Block Floating Point (BFP) [43], [45] and unstructured pruning [12], [41].

Our framework couples the SVD low-rank approximation algorithm with the streaming architecture accelerator. To evaluate the effect of the architecture in more details, we also test our low-rank approximation algorithm on SCALE-Sim [46], a single computation engine design based on systolic arrays. Ideally, halving the number of operations in a model should bring 2x speed-up in throughput. StreamSVD achieves more than 1.67x speed-up on VGG11-BN. By contrast, the speed-up on SCALE-Sim is only 1.38x because DRAM accesses increase by 29.9% after low-rank approximation. A similar

TABLE III: Comparison with other FPGA accelerators

	Model	Compression method	Post -training	Device	Clock (MHz)	DSP	FPS	Accuracy (%)	Efficiency (FPS/DSP/Clock) ($\times 10^{-9}$)
[12]	VGG-16	w16a16, pruning	no	XCZU9EG	200	1144	31	-	0.135
[41]		w16a16, pruning	no	XCZU9EG	200	1352	46	-	0.170
[42]		f16, low-rank	no	XC7VX690T	200	1728	6.58	70.46	0.019
[43]		w8a16-BFP	yes	XC7VX690T	200	1027	24.73	68.26	0.120
[18]		w16a16, low-rank	yes	XC7Z045	150	780	4.45	64.64	0.038
[44]		w16a16, low-rank	yes	XC7Z045	140	864	5.5	-	0.045
fpgaConvNet [39]		w16a16	yes	XC7Z045	125	855	5.07	-	0.047
StreamSVD		w8a16, low-rank	yes	XC7Z045	125	603	5.61	70.20	0.074
StreamSVD		w8a16, low-rank	yes	XC7Z045	125	576	7.45	65.20	0.103
[45]	ResNet-18	w2a8-BFP, low-rank	no	XC7Z020	250	202	20.48	68.23	0.406
[44]		w16a16	yes	XC7Z045	140	864	6.6	-	0.055
StreamSVD		w8a16, low-rank	yes	XC7Z045	125	576	33.77 [1]	68.39	0.469

[1] Skip connection and elementwise addition are not taken into account

conclusion is also drawn for ResNet-18, where our hardware-aware StreamSVD outperforms the systolic array architecture.

Streaming architecture avoids offloading the intermediate data between two low-rank layers to the off-chip memory and reading the data back with im2col. In addition, the streaming architecture tailors the hardware to the computation of every layer, which efficiently supports our fine-grained per-layer scheme selection. Therefore, we conclude the streaming architecture has evident advantages in mapping the low-rank approximated CNN.

F. Limitation

The 1×1 convolution, also known as pointwise convolution, has been widely used in light-weight CNN models. For example, pointwise convolutions contribute roughly 92% OPs in MobileNetV2 [47]. According to (2), pointwise convolutions in MobileNetV2 evidently favour s_0 and s_3. Unfortunately, one limitation of these two schemes is that they cannot compress convolutions with small kernel sizes. From Table I, the maximum rank (full-rank) of s_0 and s_3 is $\min(KK, C)$ and $\min(F, KK)$ respectively. When K is one, the maximum rank of both schemes is equal to one which leaves no space for compression. Therefore, decomposition schemes in Table I do not provide any speed-up for pointwise convolutions.

VII. Conclusion

This paper proposes a model-accelerator co-design framework StreamSVD. The framework considers the SVD low-rank approximation algorithm and the streaming architecture accelerator simultaneously. As a result, StreamSVD outperforms existing work on low-rank approximation by presenting the best accuracy-throughput trade-off. Future work will support more decomposition schemes such as "split-wise" [48] to compress pointwise convolutions. Apart from SVD, high-order decomposition methods including Tucker and CP decomposition will also be explored on the state-of-the-art CNN models. In addition, we will extend our work to other tasks of computer vision especially in object detection and semantic segmentation for more generality.

References

[1] "Papers with code - browse the state-of-the-art in machine learning," Accessed on: Jul. 3, 2021. [Online]. Available: https://paperswithcode.com/sota

[2] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *arXiv preprint arXiv:1803.05900*, 2018.

[3] A. Montgomerie-Corcoran and C. Savvas-Bouganis, "Def: Differential encoding of featuremaps for low power convolutional neural network accelerators," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 703–708.

[4] E. Wang, J. J. Davis, R. Zhao, H.-C. Ng, X. Niu, W. Luk, P. Y. Cheung, and G. A. Constantinides, "Deep neural network approximation for custom hardware: where we've been, where we're going," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–39, 2019.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[6] J. Zhu, Z. Qian, and C.-Y. Tsui, "Lradnn: High-throughput and energy-efficient deep neural network accelerator using low rank approximation," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 581–586.

[7] Z. Chen, Z. Chen, J. Lin, S. Liu, and W. Li, "Deep neural network acceleration based on low-rank approximated channel pruning," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 4, pp. 1232–1244, 2020.

[8] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "Fpga/dnn co-design: An efficient design methodology for 1ot intelligence on the edge," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[9] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv preprint arXiv:1712.01312*, 2017.

[10] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *arXiv preprint arXiv:1608.03665*, 2016.

[11] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 264–11 272.

[12] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on fpgas," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 17–25.

[13] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[14] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[15] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," *arXiv preprint arXiv:2001.08565*, 2020.

[16] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[17] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[18] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 26–35.

[19] J. H. Lee, S. Ha, S. Choi, W.-J. Lee, and S. Lee, "Quantization for rapid deployment of deep neural networks," *arXiv preprint arXiv:1810.05488*, 2018.

[20] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[21] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[22] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[23] J. M. Alvarez and M. Salzmann, "Compression-aware training of deep networks," *arXiv preprint arXiv:1711.02638*, 2017.

[24] J. Kossaifi, A. Bulat, G. Tzimiropoulos, and M. Pantic, "T-net: Parametrizing fully convolutional nets with a single high-order tensor," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7822–7831.

[25] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann, "Towards optimal structured cnn pruning via generative adversarial learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2790–2799.

[26] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1529–1538.

[27] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7370–7379.

[28] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 1943–1955, 2015.

[29] C. Tai, T. Xiao, Y. Zhang, X. Wang *et al.*, "Convolutional neural networks with low-rank regularization," *arXiv preprint arXiv:1511.06067*, 2015.

[30] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[31] M. Wang, B. Liu, and H. Foroosh, "Factorized convolutional neural networks," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2017, pp. 545–553.

[32] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and pattern Recognition*, 2015, pp. 1984–1992.

[33] Y. Idelbayev and M. A. Carreira-Perpinán, "Low-rank compression of neural nets: Learning the rank of each layer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8049–8059.

[34] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[35] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," *arXiv preprint arXiv:1404.0736*, 2014.

[36] Y. Li, S. Gu, C. Mayer, L. V. Gool, and R. Timofte, "Group sparsity: The hinge between filter pruning and decomposition for network compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8018–8027.

[37] A. Kouris, S. I. Venieris, M. Rizakis, and C.-S. Bouganis, "Approximate lstms for time-constrained inference: Enabling fast reaction in self-driving cars," *IEEE Consumer Electronics Magazine*, vol. 9, no. 4, pp. 11–26, 2020.

[38] S. Ribes, P. Trancoso, I. Sourdis, and C.-S. Bouganis, "Mapping multiple lstm models on fpgas," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 1–9.

[39] S. I. Venieris and C. Bouganis, "fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[40] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for fpga-based convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[41] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 9, pp. 1953–1965, 2020.

[42] C. Mei, Z. Liu, Y. Niu, X. Ji, W. Zhou, and D. Wang, "A 200mhz 202.4 gflops@ 10.8 w vgg16 accelerator in xilinx vx690t," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2017, pp. 784–788.

[43] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, "High-performance fpga-based cnn accelerator with block-floating-point arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1874–1885, 2019.

[44] P. Meloni, A. Capotondi, G. Deriu, M. Brian, F. Conti, D. Rossi, L. Raffo, and L. Benini, "Neuraghe: exploiting cpu-fpga synergies for efficient and flexible cnn inference acceleration on zynq socs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–24, 2018.

[45] Y. Chen, C. Hawkins, K. Zhang, Z. Zhang, and C. Hao, "3u-edgeai: Ultra-low memory training, ultra-low bitwidthquantization, and ultra-low latency acceleration," *arXiv preprint arXiv:2105.06250*, 2021.

[46] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[47] J. Knapheide, B. Stabernack, and M. Kuhnke, "A high throughput mobilenetv2 fpga implementation based on a flexible architecture for depthwise separable convolution," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 277–283.

[48] Y. Li, S. Gu, L. V. Gool, and R. Timofte, "Learning filter basis for convolutional neural network compression," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5623–5632.