Redundancy-reduced MobileNet Acceleration on Reconfigurable Logic For ImageNet Classification

Jiang Su, Julian Faraone, Junyi Liu, Yiren Zhao, David B. Thomas, Philip H. W. Leong, and Peter Y. K. Cheung

Imperial College London, University of Sydney
`j.su13@ic.ac.uk`

Abstract. Modern Convolutional Neural Networks (CNNs) excel in image classification and recognition applications on large-scale datasets such as ImageNet, compared to many conventional feature-based computer vision algorithms. However, the high computational complexity of CNN models can lead to low system performance in power-efficient applications. In this work, we firstly highlight two levels of model redundancy which widely exist in modern CNNs. Additionally, we use MobileNet as a design example and propose an efficient system design for a Redundancy-Reduced MobileNet (RR-MobileNet) in which off-chip memory traffic is only used for inputs/outputs transfer while parameters and intermediate values are saved in on-chip BRAM blocks. Compared to AlexNet, our RR-mobileNet has 25$\times$ less parameters, 3.2$\times$ less operations per image inference but 9%/5.2% higher Top1/Top5 classification accuracy on ImageNet classification task. The latency of a single image inference is only 7.85 ms.

Keywords: Pruning, quantization, CNN, FPGA, algorithm acceleration

1 Introduction

Modern CNNs have achieved unprecedented success in large-scale image recognition tasks. In order to obtain higher classification accuracy, researchers proposed CNN models with increasing complexity. The high computational complexity present challenges for power-efficient hardware platforms like FPGAs mainly due to the high memory bandwidth requirement. On one hand, the large amount of parameters leads to an inevitably off-chip memory storage. Together with inputs/outputs and intermediate computation results, current FPGA devices struggle to provide enough memory bandwidth for sufficient system parallelism. On the other hand, the advantages of the large amount of flexible on-chip memory blocks are not sufficiently explored as they are mostly used as data buffers which have to match with off-chip memory bandwidth. In this work, we address this problem by reducing CNN redundancy so that the model is small enough to fit on-chip and our hardware system can benefit from the high bandwidth of FPGA on-chip memory blocks.

There are existing works that have explored redundancy in CNNs on model-level and data-level separately. Model-level redundancy leads to redundant parameters which barely contribute to model computation. For example, a trained AlexNet may have 20% to 80% kernels with very low values and the computation can be removed with very limited effect to the final classification accuracy [1]. Data-level redundancy, on the other hand, refers to unnecessarily high precision for data representation to parameters. However, there are very limited work that quantitatively consider both redundancy at the same time, especially in a perspective of their impacts to a hardware system design. The contributions of this work is as follows:

- We consider both model-level and data-level redundancy, which widely exist in CNNs, in hardware system design. A quantitative analysis is conducted to show the hardware impacts of both types of redundancy and their cooperative effects.
- We demonstrate the validity of the proposed redundancy reduction analysis by applying it to a recent CNN model called MobileNet. Compared to a basedline AlexNet model, our RR-MobileNet has $25\times$ less parameters, $3.2\times$ less operations per image computation but 9% and 5.2% higher Top1/Top5 accuracy on ImageNet classification.
- An FPGA based system architecture is designed for our RR-MobileNet model where all parameters and intermediate numbers can be stored with on-chip BRAM blocks. Therefore, the peak memory bandwidth within the system can achieve 1.56 Tb/s. As a result, our system costs only 7.85 ms on each image inference computation.

About this topic, several works have explored in one perspective or another. In terms of data-level redundancy, [2] [3][4] and several other works explores FPGA based acceleration system for CNN models with fixed point parameters and activation values. But model-level redundancy is not considered for further throughput improvement. On the other side, works like [1][5] explored model-level redundancy in CNN hardware system design, but these works are presented without quantitative discussion about hardware impacts of reduced-precision parameters used in CNN models. In this work, we consider both types of redundancy and report our quantitative consideration for a MobileNet acceleration system design.

The two-level redundancy in neural networks and its impacts to hardware system design are introduced in Section 2. Section 3 introduces an FPGA system design for our Redundancy-Reduced MobileNet for ImageNet classification tasks. The experimental results are discussed in Section 4 and Section 5 finally concludes the paper.

2 Accelerating Redundancy-Reduced Neural Networks on FPGA

In this section, we firstly give a brief introduction to MobileNet [6]. Then, our redundancy reduction strategy is introduced with an explanation of how redundancy affects the hardware resource requirements. Next, we show a system architecture design for accelerating our Redundancy-Reduced MobileNet (RR-MobileNet).

2.1 MobileNet Complexity Analysis

MobileNet [6] is a recent CNN model that aims to present decent classification accuracy with reduced amount of parameters compared to CNN models with conventional convolutional (Conv) layers. Figure 1 shows the building blocks of MobileNet called depthwise separable convolutional (DSC) layer, which consist of a depthwise convolutional (DW_Conv) layer and a pointwise convolutional (PW_Conv) layer. A DW_Conv layer has a $K \times K \times N$ kernel which is essentially consist of a $K \times K$ kernel for each Input Feature Map (IFM) channel. So 2 dimensional convolutions are conducted independently in a channel-wise manner. Differently, PW_Conv layer is a special case of a general Conv layer and it has kernel size of $1 \times 1 \times N \times M$ while a general Conv layer may have kernels with a more general size of $K \times K \times N \times M$. MobileNet models, as shown in Table 2, can be formed by several general Conv layers and mostly DSC layers.

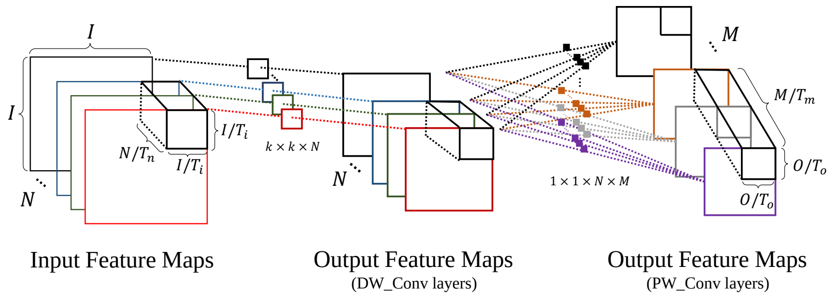

Fig. 1: Tilling in Depthwise Separable Layer for MobileNet

For a general Conv layer, below equations show the resulting operation count C_{Conv} and parameter amount P_{Conv} given that a IFM is $I \times I \times N$ and an Output Feature Map (OFM) size is $O \times O \times M$:

$$\begin{aligned} C_{Conv} &= 2 \times K^2 \times O^2 \times N \times M, \\ P_{Conv} &= K \times K \times N \times M, \end{aligned} \tag{1}$$

where 2 in Eq. 1 indicates that we consider either a single multiplication or an addition as a fundamental operation in this work. On the other side, the operation count and parameter amount of a DSC layer are as listed below:

$$\begin{aligned} C_{DSC} &= 2 \times (K^2 \times O^2 \times N + O^2 \times N \times M), \\ P_{DSC} &= K \times K \times N + N \times M. \end{aligned} \tag{2}$$

As shown in Eq.2, the amount of parameters in a DSC layer is an addition of the parameters in both DW_Conv and PW_Conv layers. In practice, a DSC layer has a parameter complexity of $O(n^3)$ while a Conv layer has $O(n^4)$ and this leads to a much smaller model for MobileNet compared to conventional CNNs [6].

2.2 Model-level Redundancy Analysis

As mentioned in Section 1, there are several works that address model-level redundancy, we use an iterative pruning strategy. Firstly, a quantization training process, which will be shortly described in Algorithm 1, is conducted on the baseline MobileNet model (Table

2). Then, an iterative pruning and re-training process is conducted. In each iteration of such process, $Prune(*)$ is applied to remove the model kernels according to β by layer-wisely thresholding the kernels values. Noticeably, our iterative pruning process is similar to strategy in [7]. However, in our strategy, a kernel is either removed or kept as a whole according to the summation of its values rather than turning it into a sparse kernel. This is called kernel-level pruning in [8]. By doing such structured pruning, we avoid using extra hardware resources to build sparse matrix formatting modules as needed in unstructured pruning strategies [5]. Finally, each pruning step inevitably leads to model accuracy loss although only less important kernels are removed. So we conduct re-training to compensate the lost model accuracy.

What pruning essentially does is changing M in equation 1 and 2 to $\beta \times M$. Correspondingly, such kernel pruning leads to a reduction of sizes of the OFM and kernels, which results in a smaller memory requirement to hardware. For example, β_l is the pruning rate of l-th layer. Kernel parameters are represented by DW_p-bit numbers while feature maps are represented by DW_a-bit numbers. For a pruned Conv layer, the memory footprint required to store kernel parameters Mem_l^p is as below:

$$Mem_l^p = K \times K \times N \times M \times DW_p \times \beta_l. \tag{3}$$

While to an SDC layer, the memory footprint is changed to following:

$$Mem_l^p = (K_l \times K_l \times N_l \times \beta_{l-1} + N_l \times M_l \times \beta_l) \times DW_p. \tag{4}$$

Eq.4 implies that the reduced parameters in the DW_Conv of a DSC layer is determined by the pruning rate of its preceding layer β_{l-1} while the the PW_Conv layer memory saving is from β_l. Specially, β_0 is 1 for the input layer.

Meanwhile, the memory footprint for storing IFMs Mem_l^I and OFMs Mem_l^O for both types of layers are as following:

$$\begin{aligned} Mem_l^I &= I_l \times I_l \times N_l \times DW_a \times \beta_{l-1}, \\ Mem_l^O &= O_l \times O_l \times M_l \times DW_a \times \beta_l, \end{aligned} \tag{5}$$

Additionally, kernel pruning also reduces computational complexity in a proportion of β. The reduced operation counts can be illustrated by Eq.1 and 2 with M displaced by its discounted value $M * \beta$ when calculating C_{Conv} and C_{DSC} separately for Conv and DSC layers.

In the next part, we will show the relationship between data-level redundancy and above-mentioned model-level redundancy as well as their cooperative effects to the hardware resources.

2.3 Data-level Redundancy Analysis

Data-level redundancy studied in this work, mainly aims to use reduced-precision parameters to replace their high-precision alternatives such as single/double-precision floating numbers that are widely used in CPU/GPU computing platforms. Instead, we explore fixed point representations with arbitrary bitwidth for parameters and activation values and quantitatively analyse their hardware impacts. Firstly, we introduce our quantization training strategy in Algorithm 2, which is used in this work for training reduced-precision neural networks.

Specially, the training procedure is completed off-line with GPU platforms. Only the trained model with reduced-precision parameters is loaded to our FPGA system for inference computation, which is the focus of this work.

Algorithm 1 Quantization Training Process for A L-layer neural network

Require: Inputs a_0, labels a^*, kernel parameters W, batch normalization parameters θ, maximum iteration number *MaxIter*, lower bound value min, upper bound value max.
Ensure: W and θ at *MaxIter* iteration.

```
for $iter = 1$ to MaxIter do
  // Forward Propagation
  $a_0^Q \leftarrow Quantize(a_0)$
  for $l = 1$ to $L$ do
    $W_l^Q \leftarrow Quantize(W_l)$
    $a_l \leftarrow layer\_forward(a_{l-1}^Q, W_l^Q)$;
    $a_l^Q \leftarrow Quantize(a_l)$;
  end for

  // Backward Propagation
  for $l = L$ to $1$ do
    $g_{a_{l-1}}, g_{W_l^Q} \leftarrow layer\_backward(g_{a_l}, W_l^Q)$
    $\theta_l \leftarrow Update(\theta_l, g_{\theta_l})$
    $W_l \leftarrow Clip(Update(W_l, g_{W_l^Q}), min, max)$
  end for
end for
```

Based on the quantization training strategies proposed in [9], we extend their training strategy to support arbitrary parameter precision as shown in Algorithm 1. In forward pass, both model parameters, or weights, W and feature map values, or activations, a are quantized before actual computations during inference. The $Quantize(*)$ function converts real values to the nearest pre-defined fixed point representation. $layer\_forward(*)$ conducts the inference computation we described in Section 2.1.

In backward propagation, parameters are updated with the gradient in terms of the quantized weights g_{W^Q} so that the network learns to do classification with the quantized parameters. However, the updating is applied to the real-valued weights W rather than their quantized alternatives W^Q so that the training error can be reserved in higher precision during training. Additionally, $Clip(*)$ helps the training to provide the quantized parameters within a particular range where values can be presented by a pre-defined fixed point representation. Concrete data representation will be introduced in Section 4. At last, we use the same hyper-parameters for training provided by [9].

Particularly, our iterative pruning and quantization training strategy (Algorithm 1) is different from the pruning and weight sharing method proposed in [7] in several ways.

Their method highlights weight sharing rather than changing the data representation. Their iterative training for pruning purposes is a separate process before weight sharing while in our approach, we do iterative pruning together with quantization training process so that model-level and data-level redundancy are both considered during training.

Above training process eventually generates a model with fixed point representations for parameters and feature map values represented with DW_p' and DW_a' bits separately. So the memory ratio between a pruned value and the its high-precision alternative are $\alpha_p = DW_p'/DW_p$ and $\alpha_a = DW_a'/DW_a$. Based on Eq.3 - 4, the memory requirement for parameters after removing both model-level and data-level redundancy is shown below for Conv layers:

$$Mem_l^p = K \times K \times N \times M \times DW_p \times \beta_l \times \alpha_p. \tag{6}$$

For DSC it is:

$$Mem_l^p = (K_l \times K_l \times N_l \times \beta_{l-1} + N_l \times M_l \times \beta_l) \times DW_p \times \alpha_p, \tag{7}$$

and for feature maps:

$$\begin{aligned} Mem_l^I &= I_l \times I_l \times N_l \times DW_a \times \beta_{l-1} \times \alpha_a, \\ Mem_l^O &= O_l \times O_l \times M_l \times DW_a \times \beta_l \times \alpha_a. \end{aligned} \tag{8}$$

We refer α as data-level memory saving factor and β as model-level memory saving factor. These two factors affects memory requirement for parameters in a multiplication way (Eq.6-8). This effect can be represented as a final saving factor of $\alpha_p \times \beta_{l-1}$ for DW_Conv and $\alpha_p \times \beta_l$ for PW_Conv and general Conv layers as shown in Eq.6 and Eq.7. Similarly, feature map values are affected by a factor of $\alpha_a \times \beta_{l-1}$ for IFMs and $\alpha_a \times \beta_l$ for OFMs as shown in Eq.8. In Section 4, we will further show that the cooperative effects of α and β are vital to our FPGA hardware architecture implementation on FPGA that provides high system performance.

3 RR-MobileNet FPGA Acceleration System Design

Based on the model-level and data-level redundancy analysis in the preceding sections, we introduce in this part what values of α and β can lead to a high-performance architecture design. In this work, we aim to achieve On-Chip Memory (OCM) storage for both parameters and feature map values. This can be achieved only with careful memory system design which is supported by a corresponding redundancy removal strategy. Firstly, we introduce the building block module design. Next, we show the conditions its memory system design should satisfy in order to implement the architecture within given FPGA resources.

3.1 System Architecture

We design a loop-back architecture, which processes our RR-MobileNet model layer by layer. Only neural network inputs, such as images, and the classification results are transferred to external of the programmable logic. So all parameters, feature maps and intermediate values are stored on FPGA OCM resources. The overall system architecture is shown in Fig.2

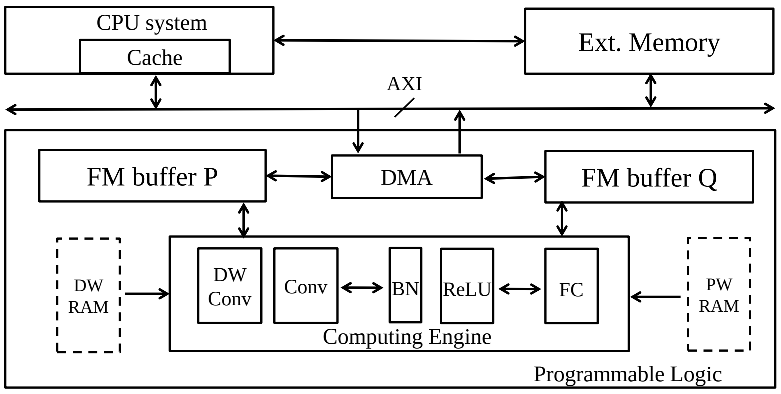

Fig. 2: System Architecture Design for RR-MobileNet

Network inputs are stored in external memory and streamed into our acceleration system by DMA through AXI bus. After Computation, the classification results are transfered back to the external memory for further usage. Within the system on the programmable logic, there are two on-chip buffers for storing feature map values. They are Feature Map (FM) buffer P and Q. Initially, the inputs from external memory are transferred to the FM buffer P and the computation can be started from this point. The computing engine module is the computational core that can process one layer at a time. Once the computing engine completes the computation of the first layer, the OFMs of the first layer will be stored in FM buffer Q. Noticeably, FM buffer P and Q are used for storage of IFMs and OFMs in an alternating manner for consecutive layers due to the fact that the OFMs of a layer are the IFMs of its following layer.

DW and PW RAMs are for parameter storage. As the module names suggested, DW RAM is for DW_Conv layer parameters while PW RAM is for ones in PW_Conv layers. There are also non-DSC layers in MobileNet structure, whose parameters are also stored in these two memory blocks. Due to the fact that DW_Conv layers have much smaller amount of parameters compared to PW_Conv layers, the DW RAM is hence used for Conv layer parameters as well as batch normalization parameters. More details about OCM utilization will be intruduced in Section 4.

The computing engine consists of DW Conv, Conv, BN and ReLU modules, which conduct the computation of either a Conv or a DSC layer. For DSC layers, its DW_Conv layer is computed by the DW Conv module followed by a BN module for batch normalization and ReLU for activation function computations. Meanwhile, its PW_Conv layer is computed in the Conv module and its following BN and ReLU modules. Due to the

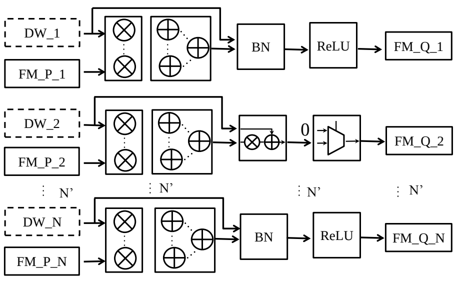

Fig. 3: PE of DW Module

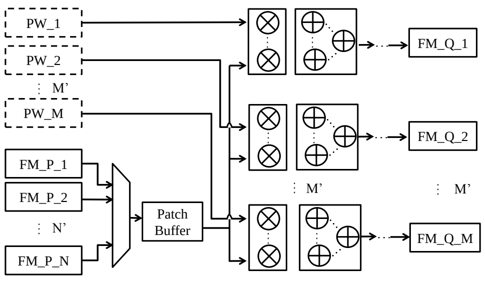

Fig. 4: PE of Conv Module

fact that PW_Conv layer is a special case of a Conv layer with all $1 \times 1 \times M$ kernels, the Conv module is also used for general Conv layer computation.

The DW module and its following BN/ReLU blocks are an array of Processing Elements(PE) as shown in Fig.3. Each PE has 32 parallel dataflow paths that are capable of processing 32 channels in parallel. As we use pre-trained batch normalization parameters, each BN module essentially takes inputs and apply a multiplication and an addition for scaling operations defined in batch normalization [10]. ReLU simply caps negative input values with 0. The Conv module is designed by conducting loop unrolling based on output feature channels M, i.e. M dataflow paths can produce outputs for the output channels in parallel. Similarly, Conv module is also consist of an array of PEs as shown in Fig.4. Each PE can produce values for 32 OFM channels in parallel. The patch buffers are used for loading the feature map numbers involved in each kernel window step and broadcasting these numbers to all computational units within the PE for OFM computations. Finally, FC modules are designed for the output layer, which essentially are parallel multiplier and adder trees for efficient computation of the last Conv layer that behaves as a Fully-connected (FC) layer.

In the next section, we specially introduce our design principles for FM buffers and parameter OCM storage mentioned in above system description.

3.2 Memory Usage

In order to avoid the time spent on off-chip memory traffic for parameters and feature map values, we aim to map both data onto the available OCMs given an FPGA device. Based on the layer-wise memory requirement analysis in Eq.6 and 7, the overall memory requirement for our Redundancy-reduced MobileNet (RR-Mobi), which contains I Conv layers and J DSC layers, is shown below:

$$\begin{aligned} Mem^{p}_{RR-Mobi} =& \sum_{i=1}^{I} K_i \times K_i \times N_i \times M_i \times \beta_i \times \alpha_p \times DW_p + \\ & \sum_{j=1}^{J} (K_j \times K_j \times N_j \times \beta_{j-1} + N_j \times M_j \times \beta_j) \times DW_p \times \alpha_p. \end{aligned} \tag{9}$$

This indicates that the memory reuse is impossible among for parameter storage among all layers in a MobileNet and the overall requirement is a summation of individual layers. Differently, the memory for feature map storage can be reused among layers because of the fact that the OFMs of layer i are only used to compute IFMs of layer $i+1$. Therefore, the memory accolated for OFM_i can be reused for storing the feature maps of the following layers. So the memory requirement for feature map storage is capped by the layer with the largest feature map values:

$$Mem^{a}_{RR-Mobi} = \bigcup_{l=1}^{l=I+J} (I_l^2 \times N_l \times \beta_{l-1} \times DW_a \times \alpha_a + O_l^2 \times M_l \times DW_a \times \beta_l \times \alpha_a), \tag{10}$$

where $\bigcup_{l=1}^{l=I+J}$ returns the maximum memory requirement for feature maps of any single layer among all $I+J$ layers. If Mem_{OCM} represent the OCM resources available on a particular FPGA device, below condition should be valid in a memory system design:

$$Mem^{a}_{RR-Mobi} + Mem^{p}_{RR-Mobi} < Mem_{OCM}. \tag{11}$$

So our redundancy removal strategy should ideally provide values of α and β for each layer that satisfies Eq.11. In the experiments in Section 4, we will show our resulting strategy of redundancy removal for above-mentioned purposes.

3.3 Layer Tilling

As introduced in section 3.1, feature maps are organized based on channels for parallel access. However, some layers have just a few channels but with large amount of numbers in each channel or the other way around, which lead to an efficient storage. For example, the IFMs of the first Conv layer in MobileNet can be images in the ImageNet dataset and the size can be 224×224×3. However, our proposed system architecture would have higher efficiency if the feature maps have many channels but only reasonably small amount of numbers in each, like the DSC hidden layers in the middle part of the MobileNet topology. So we do tilling to balance the memory for layers like the first Conv layer by divide the IFMs along the channel direction and break them down to smaller blocks that can fit the memory system more efficiently.

An example of applying tilling to a DSC layer is shown in Fig.1. $I \times I \times N$ IFMs are tilled into $T_i \times T_i \times T_n$ components, which can be computed in parallel in the computing engine. Similarly, for the PW_Conv layer, OFMs can be divided into $T_o \times T_o \times T_m$ parallel tiles. In the Section 4, we will show our tilling strategy for our RR-MobileNet system.

4 Experimental Evaluation

4.1 Experimental Results

In this section, we report implementation details of our RR-MobileNet redundancy removal strategy and acceleration system. Meanwhile, we compare our restuls with several other works to demonstrate the validity of our proposed method.

4.2 Experimental Settings

The system is implemented with Vivado HLS complied in Xilinx SDx v2016.3 version. The working clock frequency is 150 MHz. We use Xilinx Zynq UltraScale+ MPSoC as our design hardware platform. Its quad-core ARM Cortex-A53 and a XCZU9EG FPGA chip are separately corresponding to the CPU and programmable logic in Figure 2. We train and predict our RR-MobileNet model on ImageNet dataset and report the results from predicting on the validation set for the Top1/Top5 classification accuracy. The overall system resource utilization is shown in Table 1. Our system is mainly limited by on-chip memory resources which are used for storing all quantized parameters and feature maps. Considering the data access based on Eq.11, the resulting number of PEs

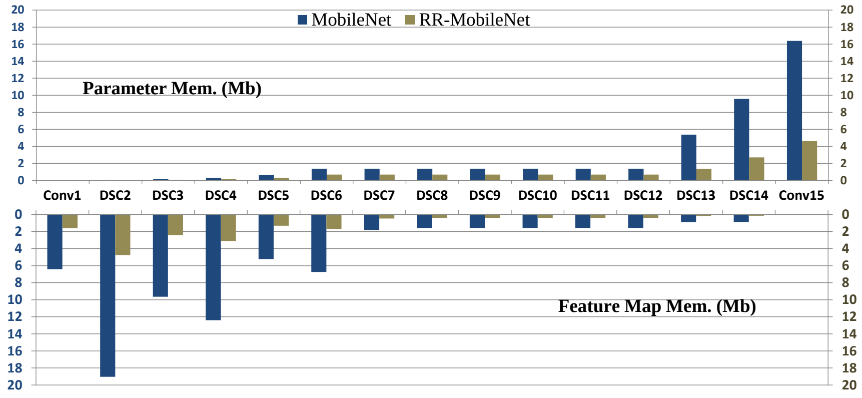

Fig. 5: Per-layer Memory Requirement With/Without Redundancy Removal

in DW_conv and Conv moduls are 9. Therefore, up to $9 \times 32 = 288$ kernel channels can be computed in parallel. According to this information, the final RR-MobileNet topology and our redundancy removal strategy are described in Table 2. Meanwhile, a resulting layer-wise memory requirement before and after pruning is shown in Fig.5. It is shown in the figure that the layers towards the input and output layers are both memory-heavy layers but for different reasons. The input side layers are because of massive feature maps numbers while the output side layers are due to the large amount of parameters. In order to handle this, we apply tilling especially with higher factors to these layers as shown in Table 2. Meanwhile, we apply β to the hidden layers in the middle to further bring down the memory requirements. Specially, our design finally uses a α_a of 0.25 and α_w of 0.5 for data representation. we found this α and β combination gives us fittable design with reasonable model accuracy presented. So the determined data representations for parameters and feature maps are Q2.6 and Q2.2 fixed point values with setting min and max in Algorithm 2 to -1 and 1 separately. We adopt TensorFlow implementation running on an Nvidia Titan X GPU for iterative pruning and quantization training. Only inference is accelerated on FPGA. The final classification accuracy is shown in Table 3.

As shown in Table 3, we compare our RR-MobileNet with several other CNN hardware accelerators as well as the baseline MobileNet without any redundancy reduction.

Table 1: Resource Utilization

	BRAM_18K	**DSP**	**FF**	**LUT**
Usage	1729	1452	55K	139K
Total	1824	2520	548K	274K
Util.	95%	58%	11%	51%

Table 2: Redundancy Reduction and Tilling

	Conv1	**DSC2**	**DSC3**	**DSC4**	**DSC5**	**DSC6**	**DSC7-12**	**DSC13**	**DSC14**	**Conv15**
M	32	64	128	128	256	256	512	1024	1024	1000
Pruned M	32	64	128	128	256	256	288	576	768	1000
β	0	0	0	0	0	0	0.56	0.56	0.75	0
Tilling (Ti/Tj/Tn/Tm)	4/4/1/1	2/2/1/1	2/2/1/1	1/1/1/1				1/1/1/2	1/1/1/3	1/1/1/4

It shows that our re-training strategy can reserve the original model accuracy to some extend while reducing redundancy from the neural network. Our RR-MobileNet removes 42% of the original MobileNet parameters. Specially, compared to the baseline AlexNet design, our model requires 25× less parameters and 3.2× less operations per image computation to achieve about 9% and 5.2% higher Top1/Top5 accuracy.

Table 3: Comparison With Other Works

Work	**Model**	**Params (M)**	**GOps/F**	**FPGA**	**Freq. (MHz)**	**Prec. (W/a)**	**#ms per Inf. (ms)**	**GOPS**	**Top1 Acc.**	**Top5 Acc.**
[2]	AlexNet	62.4	2.27	Stratix-V	100	8/10	$\sim$13	114.5	55.6	79.3
[4]	VGG-16	132	30.94	Zynq XC7Z045	150	16/16	224.6	136.97	68	87.9
[6]	MobileNet	4.2	1.13	-/-	-/-	16/16 (FP)	-/-	-/-	0.71	0.90
Ours	RR-MobileNet	2.5	0.72	XCZU9EG	150	8/4	7.85	91.2	64.6	84.5

We also compare our results with several other state-of-the-art works as shown in Table 3. Because our work is the first one that maps MobileNet on FPGA platforms, we can only compare with existing works that focus on different CNN models. Our system can achieve lower latency than other works mainly because of the high on-chip memory bandwidth access, which can achieve a top 15.6 Tb/s. However, current memory system design also limits the PE parallelism due to fixed data access pattern. So the throughput is relatively lower than other works. This will be studied in our future works.

5 Summary and Conclusion

In this work, we present our quantitative analysis of both model-level and data-level redundancy in MobileNet and implement an FPGA acceleration system for our RR-MobileNet. Although we only focus on MobileNet model in this work, our redundancy removal methods are general to a broad range of neural networks. In the future work, we will focus on improving our system throughpout using orthogonal CNN loop manipulation techniques. We will also extend our study to other types of neural networks.

Acknowledgments

The authors from Imperial College London would like to acknowledge the support of UKs research council (RCUK) with the following grants: EP/K034448, P010040 and N031768. The authors from The University of Sydney acknowledge support from the Australian Research Council Linkage Project LP130101034.

Bibliography

[1] A. Parashar and et. al., “SCNN: an accelerator for compressed-sparse convolutional neural networks,” *CoRR*, vol. abs/1708.04485, 2017.

[2] Y. Ma and et. al., “Scalable and modularized rtl compilation of convolutional neural networks onto fpga,” *FPL*, 2016.

[3] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. H. W. Leong, M. Jahre, and K. A. Vissers, “FINN: A framework for fast, scalable binarized neural network inference,” *CoRR*, vol. abs/1612.07119, 2016.

[4] J. Qiu and et. al., “Going deeper with embedded FPGA platform for convolutional neural network,” in *Proc. ACM/SIGDA ISFPGA*, pp. 26–35, 2016.

[5] S. Han and et. al., “EIE: efficient inference engine on compressed deep neural network,” 2016.

[6] A. Howard and et. al., “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *CoRR*, vol. abs/1704.04861, 2017.

[7] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman coding,” *CoRR*, vol. abs/1510.00149, 2015.

[8] S. Anwar and et. al., “Structured pruning of deep convolutional neural networks,” *CoRR*, vol. abs/1512.08571, 2015.

[9] 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*, vol. abs/1606.06160, 2016.

[10] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proc. ICML*, pp. 448–456, 2015.