SqueezeVGGNet: A Methodology for designing low complexity VGG Architecture for Resource Constraint Edge Applications

Ramesh Reddy Chandrapu, Chandrajit Pal, Anagha Tilak Nimbekar, Amit Acharyya
ee17resch01007@iith.ac.in, palchandrajit@gmail.com, {ee20resch01001, amit_acharyya)@iith.ac.in
Electrical Engineering Department, IIT Hyderabad, India

***Abstract*—Convolutional Neural Networks also known as ConvNets are now extensively used in various machine learning tasks for solving various problems in computer vision, biomedical, defence industry, entertainment etc. These neural networks for most of the applications are focused towards increasing the accuracy. However, besides maintaining the accuracy within a tolerable range, reduction in the network model size can have a lot of advantages from its mobility, easy deployment, remote upgradation and energy efficiency point of view. To attain these advantages, we propose a universal strategy to realize the convolution operation of a n x n filter kernel with fewer parameters, which also reduces the number of channels. We have proposed a compressed VGGNet model based on VGGNet neural network which resulted in 20x lesser parameters compared to its classical counterpart with an improved inference time by 3 times whilst maintaining similar accuracy. A complete hardware design of the compressed VGG architecture has also been implemented. A quantitative and qualitative analysis for various variants of VGGNet and other models reveal the reduction in the number of parameters in the range of 18-20x and the number of network operations contributing to the model complexity has shown a reduction of 2.5x with respect to its vanilla counterpart making it easier to deploy onto FPGAs and edge devices**

***Keywords*—squeezeVGGNet, CNN (Convolutional Neural Network), compaction, expansion, COMPEXP**

I. INTRODUCTION

Over the past decade, there has been an extensive research and development on machine learning algorithms applied to computer vision based applications. Much of its success and rising popularity can be attributed to the ground breaking results achieved by the convolutional neural networks (CNNs) or convnets like Alexnet, ZF Net, googLeNet, VGGNet etc on Imagenet Large Scale Visual Recognition Challenge [10], 2012-13 [1]. Convnets are getting difficult to deploy on resource constraint mobile Edge platforms due to their increasing model size. The parameters of a neural network are typically the weights of the connections that are learned during the training stage that adds to the model size.

Therefore, to widely deploy on resource constrained edge devices, the number of parameters constituting the convnet model needs to be optimized to operate on low power edge devices. The challenge lies in optimizing the network model architecture while maintaining similar accuracy. The purpose of this work is to introduce an architecture which will reduce the model size of VGGNet by reducing the number of parameters without affecting the classification accuracy.

To achieve this, a deep neural net should be taken and be compressed in a lossy manner, upto an extent which will not drop the performance below a tolerable range.

Our main contributions are highlighted as follows:

• We propose an end-to-end compressed VGGNet architecture and coined it as squeezeVGGNet (VGG for the edge devices) with reduced the number of parameters consisting of compaction and expansion modules containing combination of axa, bxb and 1x1 convolution architectures, where a and b are almost half of kernel resolutions. The number of input channels fed to each layer has also been decreased to reduce further the model size.

• Our proposed methodology has been experimented on benchmarks nets like VGGNet, miniGoogleNet and has resulted in a reduction of approximately 20X parameters compared to its classical counterpart with 3 times improved inference speed.

• With an aim to design a deep neural network for resource constrained and low power edge devices, we have designed squeeze VGGNet hardware accelerator on FPGA corresponding to our proposed algorithm. The rest of the paper is divided into the following sections.

Section II contains a brief background. Section III describes proposed methodology and the hardware design accelerator. Section IV contains the experimental results and analysis. Section V concludes with scope of its future extension.

II. BACKGROUND

A. Network compaction

Recent publications have reported the initiative taken by various researchers on network model compression. Clever approaches started with applying singular value decomposition (SVD) on deep networks, followed applying pruning, quantization and Huffman coding on deep networks by Song Han et al. [2].

B. Compression through network rearchitecture

Convolutional deep neural networks are being used for the past two decades. It all started with the early work by LeCun et al. [3] using the uniform 5x5xNo of Channels. The VGG network architecture was introduced by Simonyan and Zisserman [4], the runner up at the ILSVRC 2014 challenge. It is different from the winners of ILSVRC 2012 and 2013 performing only 3x3 convolutions over the entire network. Few models like Network-in-Network [5] and the GoogLeNet moodel architectures [7][8][9] use 1x1 filters in its layers. Residual Networks (ResNet) [6] and Highway

Networks [11] were the first to propose the use of connection skip over multiple layers, also known as bypass connections. This delivered an additional 2 percentage-point improvement on Top-5 ImageNet accuracy unlike the non-bypassing one. Squeezenet [12] achieves Alexnet level accuracy on Imagenet (both top-1 and top-5) with 50x fewer parameters

III. PROPOSED METHODOLOGY

In this paper, a generic compression scheme for CNN architecture is presented. It consists of nxn filters of any size so that they have fewer parameters while maintaining similar accuracy. We have implemented this on VGG and achieved new architecture compressed VGG whose size is reduced by 20x times with a negligible drop in the prediction accuracy. Let's consider a generic convolution of resolution nxnxf size,where nxn is the filter kernel resolution and f is the number of such filters. We plan to reduce this size by using smaller convolution filter kernel resolution together with less number of filters. We do this by –

- **Choosing compact and expand parameters a and b:**

  We choose axa and bxb convolutions such that both a and b are less than n and roughly close to n/2.

  $$axa + bxb < nxn$$

- **Reduction in the number of input channels to nxn filters**: Considering a convolution layer that has nxn filters. The total number of parameters in this layer is (no of input channels)x (no of filters) x (nxn). We decrease the number of input channels to (nxn) filters by using compexp layer, as shown in architecture of Fig. 1.

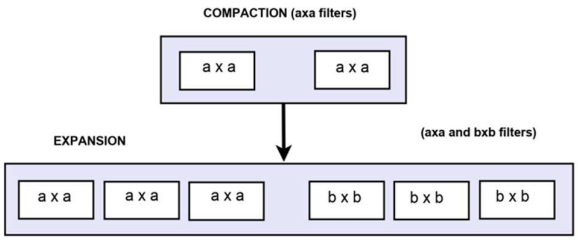

Fig 1: compexp module

Our compexp module is applicable for any nxn convolution filter. We have three tunable parameters - c_{axa}, e_{axa}, e_{bxb}. The number of filters in the compaction layer i.e c_{axa} is always kept lower than the combined total number of filters in the expand layer. i.e Number(c_{axa}) < Number (e_{axa} +e_{bxb}) so as to reduce the number of input channels to the nxn filters.
The main basis of selecting a and b is –

$$a = (n + 1) / 2 - 1 \text{ if } (n+1) / 2 \text{ is even} \quad (1)$$
$$= (n + 1) / 2 \quad \text{if } (n+1) / 2 \text{ is odd} \quad (2)$$
$$b = (n + 1) / 2 + 1 \text{ if } (n+1) / 2 \text{ is even} \quad (3)$$
$$= (n + 1) / 2 \quad \text{if } (n+1) / 2 \text{ is odd} \quad (4)$$

c_{axa} is the number of (axa) filters in the compaction layer, e_{axa} is the number (axa) filters in the expand layer and e_{bxb} is the number of (bxb) filters in the expand layer.
Let us compare the parameters in both cases:

A. Traditional Approach

Total number of parameters in a layer is

$$n \times n \times \text{filters} \times cr \times \text{input channels} \quad (5)$$

cr is the compression ratio and is defined as the ratio of the number of filters in the compaction layer to the total number of filter in the compaction and expand layer.

B. Our Approach

Number of parameters in Compaction Layer is

$$((n+1) / 2 -1)^2 \times \text{filters} \times cr \times \text{input\_channels} \quad (6)$$

Number of parameters in Expand Layer is

$$((n+1)/2 - 1)^2 \times \text{filters} \times (b/a+b) \times \text{input\_channels} + ((n+1)/2 + 1)^2 \times \text{filters} \times (a/a+b) \times \text{input\_channels}. \quad (7)$$

Where a = (n+1)/2-1 and b = (n+1)/2+1

For example, let's take 7x7 64 filters and number of input channels be 64.

Number of parameters by following the traditional approach in equation 5= 7x7x64x1x64 = **200704**

Our approach:

- compaction layer parameters = $(8/2 - 1)^2$ x 64 x 0.25 x 64 = **9216**
- The 0.25 in the compaction layer is 2/8 (since the ratio of the number of filters in the compaction layer to the total no of combined filters in compaction and expand layer).
- Similarly Expand layer parameters = **15360**

Here a = 3 and b = 5. Total number of parameters = (9216 + 15360)= **24576**. Therefore, it can be clearly observed that the number of parameters has reduced drastically. This extends the compression to higher size filters. Similar is the scenario for filters with resolution 5x5.

In the case of (n+1) / 2 is odd, only one type of filter resolution is used in the expansion layer following equation (3). For example for 5x5x64x64 convolutional layer (where n=5). Here the parameters would be 102400=(5x5x64x64). Now, computing by our methodology we get, in the compaction layer as (5+1)/2 is odd, it will have 3x3x16x64 parameters in compaction layer and in expansion layer 2x3x3x16x64 which would amount to = **27648**. This is very less again in comparison to **102400**.

Complete architecture of Squeeze VGG is given in Table II and in Fig. 2. For any CNN architecture, we employ the following methodology for compression:

- **Leave the first layer unpruned:** In any CNN architecture, images fed to the first layer have 3 channels (RGB). If the first layer has 64 such 7x7 convolution filters, number of parameters would be (64x7x7x3) =9408. If we apply compaction module on the first layer, the number of parameters would increase to (3x3x3x16 + 16x5x5x24 + 16x3x3x40) =15792.
- Replace nxn filters with compexp modules: Majority of the filters will be replaced by axa and bxb where a and b are less than n. For example, 7x7 convolution filter will be replaced by 3x3 and 5x5.

C. Proposed Hardware Design Flow

To implement the machine learning Inference on the custom boards based on the Zynq Ultrascale+ FPGA, we designed the RTL architecture for the compressed VGGNet inference

accelerator as shown in Fig. 3. The hardware modules are labelled alphabetically. The proposed architecture comprises of four major components namely the master controller, storage memory, convolutions and interfaces. The master controller consists of five parts like (i) Channel controller(c), (ii) Memory controller(d), (iii)Address controller(e), (iv) Layer decision block(f) and (v) DDR controller(g). Each controller controls and monitors various sub modules in the process of execution of the net. These sub modules includes convolutions(a,b), Bias addition(q), PRelu activation(h), pooling modules (j), Fully connected layer(m) respectively. The dataflow among the various modules of the Squeeze VGG is shown in Fig. 3 with sequence numbers.

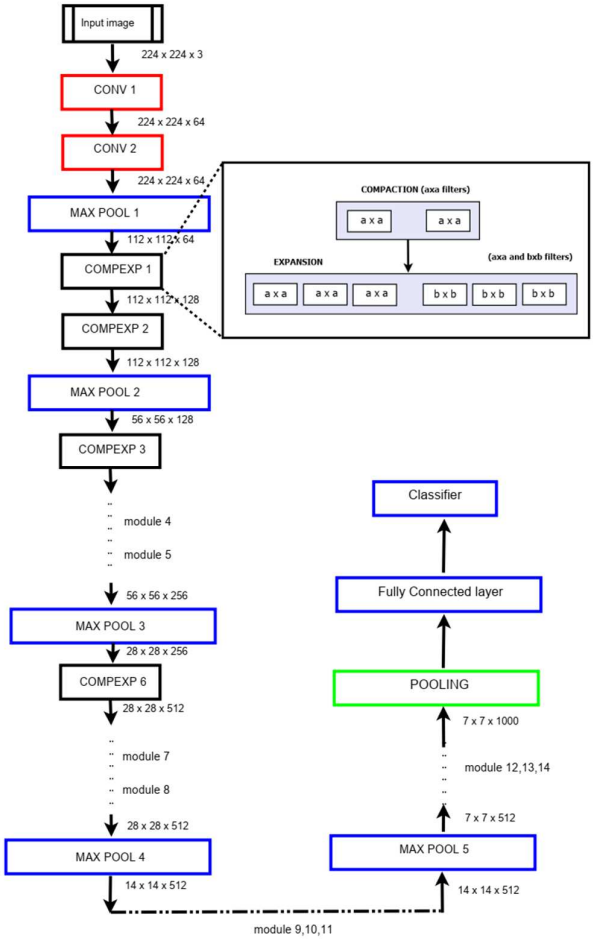

Fig 2: Macroarchitectural view of the squeeze VGGNet architecture

IV. EXPERIMENTATL RESULTS

We have conducted our experiments using Python and caffe deep learning frameworks on Intel processor Core i7-880 @3.20GHz with Linux Ubuntu 16.04 operating system. The training was performed on 2 Tesla K40 Nvidia GPUs. The proposed methodology is applied to some of the popular nets to test our hypothesis. Experimented on VGG-16 and designed a compressed VGG-16 with a reduction of 18-20x in size and 3x in inference speed with some minimal loss in accuracy in comparison with the classical VGG model.

Macroarchitectural view of Squeeze-VGG is shown in Fig. 2 and Table II. To extend our COMPEXP modules to larger convolution filters, for experimentation purpose, we chose to further optimize the architecture and called it as VGG-F architecture. It has 7x7 convolution filters in the first layer followed by 5x5 in the second layer. We compressed the 5x5 convolution filters by using 3x3 convolution filters in both compaction and expansion layer. To ensure that the output of axa and bxb convolution layers in the expansion layer have the same size, zero padding is used when a < b.

Convolution layers are used instead of fully connected layers, inspired by Network in Network architecture [5]. A batch size of 128 was used with a learning rate starting at 0.01 and decayed it with momentum of 0.1 every 10,000 iterations. A total of 1000000 iterations were performed.

Nvidia embedded GPU module-Jetson TX2 was used as an embedded board to deploy our compressed models. Inference time of 30 ms is achieved using SqueezeVGGNet compared to 90ms without compression on Jetson-Tx2 board. We applied our compression methodology on VGG-16 and VGG-F trained on ImageNet dataset with performance results shown in Table III.

We applied our compression methodology on MiniVGGNet and miniGoogleNet trained on CIFAR 10 dataset. Our COMPEXP modules can be applied on nxn filters, which resulted in compressing all the benchmark neural nets, which has 5x5 convolution filters. The training time and complexity of both the networks was reduced with the help of our COMPEXP modules. Table III compares the accuracy, model size and number of operations of the compressed version of the networks with our proposed methodology with respect to their classical counterparts.

A quantitative and qualitative analysis reveals that the number of parameters (contributing model size) has been reduced in the range of 18-20x in size and the number of operations contributing to the model complexity has shown a reduction of 2.5x. In addition to decrease in inference time, model size and number of parameters, our networks also reveals a drastic reduction in the number of operations, making it easier to deploy onto FPGAs. Our proposed SqueezeVGGNet is simulated and synthesized on zynq ultrascale+MPSoC on zcu-102 board using CIFAR-10 dataset. The FPGA resource utilization summary is shown in Table I.

V . CONCLUSION

There is a general saying that the deeper we go the better we can visualise. We feel the same way for compression of these nets. Our proposed squeezeVGGNet model based on VGGNet neural network resulted in 20x lesser parameters compared to its classical counterpart with an improved inference time by 3 times whilst maintaining almost similar accuracy. A complete hardware design of the compressed VGG architecture has also been implemented. A quantitative and qualitative analysis for various variants of VGGNet and other models reveal the reduction in the number of parameters in the range of 18- 20x and the number of network operations contributing to the model complexity has shown a reduction of 2.5x with respect to its vanilla counterpart making it easier to deploy onto edge devices. Our future plans to decompose the any nxn to components of 1x1 and 3x3 by repetitive use of our current method. We also have plans to extend this compression from

classification to the fields of object detection and segmentation. We are also currently trying to combine mobile nets into this architecture as well to give both speed and light weight capabilities to the current benchmark nets.

Table I: FPGA utilisation on Xilinx Zynq Ultrascale+ MPSoC on Zcu-102 board

Resource	Estimation	Available	Utilization
LUT	246233	274080	89.84
LUTRAM	4480	144000	3.11
FF	264825	548160	48.31
BRAM	85.50	912	9.38
I/O	48	328	14.63
BUFG	1	404	0.25

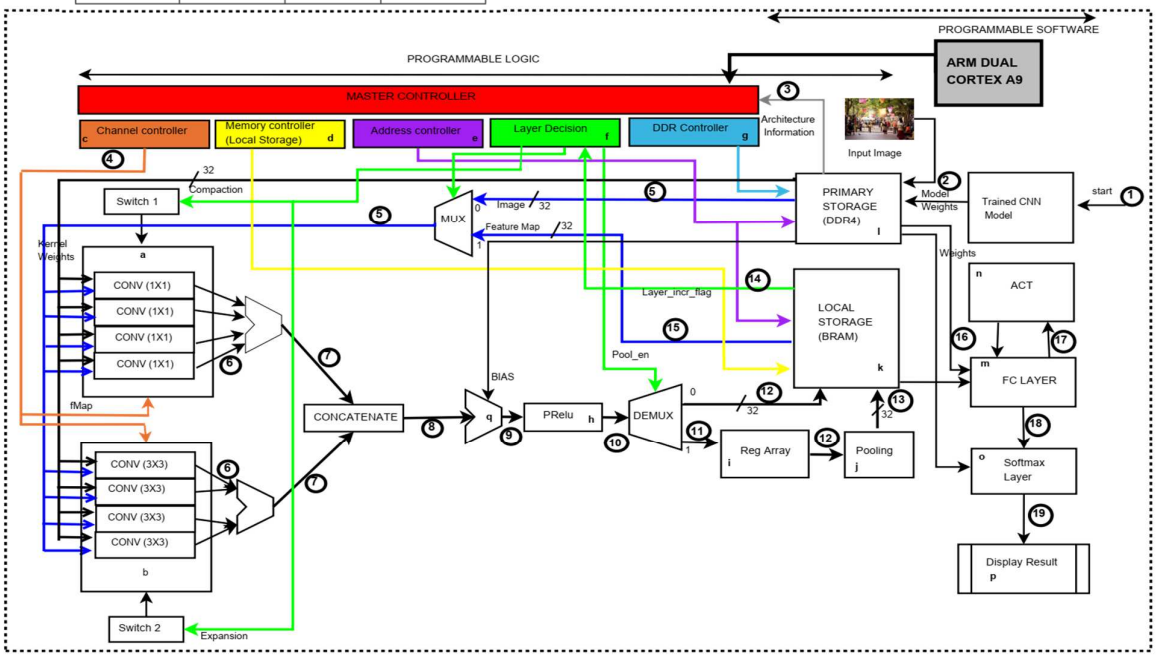

Fig 3: Architectural design of compressed VGG neural network inference accelerator on FPGA [13]

TABLE II: Proposed SqueezeVGGNet architectural dimensions

Layer	Output size	c_{axa} (squeeze)	e_{axa} (expand)	e_{bxb} (expand)	# parameters (Original VGG)	# parameters (squeezeVGGNet (Our proposed))
Input image	224x224x3					
Conv1	224x224x64				1.79k	1.79k
Conv 2	224x224x64				36.93k	36.93k
Max pool 1	112x112x64					
COMPEXP 1	112x112x64	16	64	64	73.86k	11.41k
COMPEXP 2	112x112x64	16	64	64	147.58k	12.43k
Max pool 2	56x56x128					
COMPEXP 3	56x56x128	32	128	128	295.17k	45.34k
COMPEXP 4	56x56x128	32	128	128	590.08k	49.44k
COMPEXP 5	56x56x128	32	128	128	590.08k	49.44k
Max pool 3	28x28x256					
COMPEXP 6	28x28x256	64	256	256	1.18M	0.1808M
COMPEXP 7	28x28x256	64	256	256	2.36M	0.197M
COMPEXP 8	28x28x256	64	256	256	2.36M	0.197M
Max pool 4	14x14x512					
COMPEXP 9	14x14x512	64	256	256	2.36M	0.197M
COMPEXP 10	14x14x512	64	256	256	2.36M	0.197M
COMPEXP 11	14x14x512	64	256	256	2.36M	0.197M
Max pool 5	7x7x512					
Conv 12	7x7x1024				102.76M	4.72M
Conv 13	7x7x1024				16.78M	1.05M
Conv 14	7x7x1020				4.1M	1.02M
Avg pooling	1x1x1000					
Total -					138.36M	8.17M

Table III: Comparison of Top 5 accuracy by executing on Imagenet dataset for VGG-16, VGG-F and, CIFAR10 dataset for MiniVGGNet and MiniGoogleNet

Architecture	Original ->Compressed Top - 5 accuracy	Original ->Compressed model size	Original ->Compressed # operations	Original ->Compressed # parameters
VGG - 16	91.1 ->86.3	548 mb ->29 mb	15.47G ->3.42 G	138.36M ->8.17M
VGG - F	83 ->80	248 mb ->15 mb	7.26G ->5.39 G	60.83M ->2.96 M
MiniVGGNet	81 → 80	8.7 MB → 0.3 MB	26 M → 12 M	2168.362 K → 95.674 K
MiniGoogLeNet	85 → 82	6.8 MB → 2.2 MB	234 M → 116 M	1656.250 K → 555.146 K

ACKNOWLEDGEMENT

The authors like to specially thank the contribution of Mr Srikar YM and Sai Niranjan Reddy for their ideas and constant support in making this project successful. This project is partially funded by Defence Research Development Organization. The CAD tools and reconfigurable platforms are supported by Ministry of Electronics and Information Technology, Govt of India.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (June 2017), 84–90. https://doi.org/10.1145/3065386

[2] Song Han and Huizi Mao and William J. Dally, Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, 2016, 10.48550/ARXIV.1510.00149

[3] Lecun, Y. and Bottou, L. and Bengio, Y. and Haffner, P., Proceedings of the IEEE, Gradient-based learning applied to document recognition,1998,volume=86,number=11, pages=2278-2324

[4] Karen Simonyan and Andrew Zisserman,Very Deep Convolutional Networks for Large-Scale Image Recognition,2015, 10.48550/ARXIV.1409.1556.

[5] Min Lin and Qiang Chen and Shuicheng Yan, Network In Network, 2014, 10.48550/ARXIV.1312.4400.

[6] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[7] C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2818-2826, doi: 10.1109/CVPR.2016.308.

[9] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. 2017. Inception-v4, inception-ResNet and the impact of residual connections on learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI'17). AAAI Press, 4278–4284.

[10] J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

[11] Rupesh Kumar Srivastava and Klaus Greff and Jurgen Schmidhuber, ¨ Highway Networks, 2015, 10.48550/ARXIV.1505.00387.

[12] Forrest N. Iandola and Song Han and Matthew W. Moskewicz and Khalid Ashraf and William J. Dally and Kurt Keutzer, SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model. size, 2016, 10.48550/ARXIV.1602.07360.

[13]ZYNQ FPGA, xilinx.com/support/documentation/sw manuals/ xilinx2019 2/ug1165-zynq-embedded-design-tutorial.pdf