A NOVEL METHOD TO REGENERATE AN OPTIMAL CNN BY EXPLOITING REDUNDANCY PATTERNS IN THE NETWORK

Sirish Kumar Pasupuleti, Narasinga Rao Miniskar,Vasanthakumar Rajagopal, Raj Narayana Gadde

Samsung R&D Institute India - Bangalore Pvt. Ltd. Bangalore, India
(sirish.p, nr.miniskar, vasanth.r, raj.gadde)@samsung.com

ABSTRACT

Deploying Convolution Neural Networks (CNN) based computer vision applications on low-power embedded devices is challenging due to massive computation and memory bandwidth requirements. Research is on-going on faster algorithms, network pruning, and model compression techniques to produce light-weight networks. In this paper, we propose a novel method which exploits a redundancy pattern in the network to regenerate an efficient and functionally identical CNN for a given network. We identify the pattern based on the layer parameters (kernel size and stride) and data flow analysis among the layers to avoid the redundant processing and memory requirements while maintaining identical accuracy. Our proposed method augments the state-of-the-art pruning and model compression techniques to achieve further performance boost-up. The proposed method is experimented with the Caffe [1] framework for ResNet-50 [2] inference on Samsung smartphone with an octa-core ARM Cortex-A53 processor. The results show an improvement of 4x in performance and memory at layer level, $\sim$22% performance improvement and 6% memory reduction at network level.

Index Terms— Deep Neural Networks, Convolution Neural Networks, ResNet, Caffe, light-weight network

1. INTRODUCTION

In recent years, CNNs are playing a key role in the field of computer vision [3, 2] for image classification, object detection, etc. The CNN based applications are getting deployed into smartphones, drones, etc. However, achieving the real time performance on embedded platforms is a challenge due to massive memory accesses and computations. The Deep Neural Networks (DNN) are getting complex and more deeper [2], which makes it even harder to deploy these networks on mobile devices.

Current research is majorly focusing on achieving the real time performance on embedded platforms with faster algorithms, network pruning and model compression techniques [4, 5, 6]. Andrew et al. [7] proposed fast algorithms for convolution neural networks using Winograd's minimal filtering algorithms. Song et al. [8] proposed a three stage pipeline containing pruning, quantization and Huffman coding stages to compress the networks which produces 3x to 4x layerwise speedup. On-the-fly connection pruning techniques to reduce the network complexity is proposed by Yiwen et al. [9]. Jiaxiang et al. [10] proposed a quantized CNN framework, which results in 4x speedup for VGG-16 [11] network. A technique to prune the filters which does not result in sparse connections is proposed by Hao Li et al. [12], improves the inference performance by 38% for ResNet-110 [2]. Most of these techniques produce highly compressed light-weight networks with a slight compromise on accuracy levels. Hence, there is a trade-off between amount of compression and accuracy levels. In the state-of-the-art, the optimization of the network by exploiting the redundancy patterns without affecting the accuracy is not yet explored as per our knowledge.

In this paper, we propose a novel method to generate an optimal and functionally identical network by analyzing the layer parameters and data flow patterns among the layers in the network. In particular, the method analyses kernel size and stride among the layers to identify redundancy patterns and to move the stride to prior layers to avoid redundant processing. The proposed method does not alter the behaviour of the original network. Hence, the original functionality and accuracy are unchanged. The method can be seamlessly integrated as a pre-processing step with any CNN framework for the on-the-fly network optimization. The computation complexity of our method is $O(M^2)$, where M is total number of layers and this complexity is negligible when compared to any layer computation. However, it can be used as a off-line standalone tool to be applied once for each network.

The proposed method produces an optimal network by exploiting the redundancy patterns in an unique way which is independent of the state-of-the-art pruning and model compression techniques. Hence, the proposed method can also be applied to exploit the redundancy patterns on already pruned and compressed networks to augment the performance. The proposed method is integrated with Caffe framework for ResNet-50. The results have shown $\sim$22% performance improvement and 6% memory reduction with the optimal network generated by our method. The method can be utilized for other CNNs if the redundancy pattern is present. The exploration of more redundancy patterns in CNNs is a future work.

2. RESIDUAL NETWORK

Kaiming et al. [2] proposed the deep Residual Network (ResNet), which has achieved 3.57% top-5 error on the ImageNet test set and won the 1st place in the ILSVRC-2015 [13] classification task. The ResNet became very popular and many researchers are adapting it as the base network to solve various computer vision tasks.

The ResNet has been built by stacking residual blocks, two such residual blocks are shown in Fig. 1(a). Each residual block is represented as a group of basic building blocks and each basic building block is a combination of various layers. The basic building block starts with either a Convolution (Conv) or an Element wise (Eltws) layer, and comprises all the subsequent layers until the next Conv or Eltws layer. The first building block contains the Eltws and ReLU layers. The subsequent building blocks start either with Conv or Eltws layers, and followed by a Batch Normalization (BN), scaling and ReLU layers. The layers are shown with a stride parameter which is a shift required to select the next pixel for processing in the image. A stride value of 1 means all the pixels are processed in the image and stride value 2 means every alternative pixel processing is skipped both in horizontal and vertical directions which results in processing only one fourth of the total pixels in the image.

Each residual block has two independent paths, **normal path** and **skip path**. The *normal* path contains a series of building blocks. The *skip* path formed between two Eltws building blocks, may have a direct connection or may contain intermediate building blocks.

3. PROPOSED METHOD TO REGENERATE OPTIMAL NETWORK

Our method analyses data flow in the network to identify specific patterns which can be exploited to avoid the redundant processing. We have identified a redundancy pattern based on convolution kernel size 1x1 with stride > 1 in ResNet-50. Since convolution 1x1 processing doesn't have any dependency on the neighboring pixels and stride > 1 skips the processing of pixels present at positions non-multiple of stride value, these redundant pixels can be excluded from processing in the prior layers. If the redundancy pattern is found in a network, our method checks the possibility of moving the stride to the prior layers to avoid the redundant processing.

As shown in Fig. 1(a), all the layers from Eltws-A to ReLU-C has a stride value of 1 except Conv4 and Conv7 layers which has a stride value of 2 in both horizontal and vertical directions. Conv4 and Conv7 layers process only pixels at positions multiple of stride value 2 and remaining 75% pixels are ignored from further processing in the network. Hence, the stride 2 can be moved to farthest possible prior layer to avoid processing of the redundant pixels. The stride value 2 can be moved to immediate parent layer of both Conv4 and

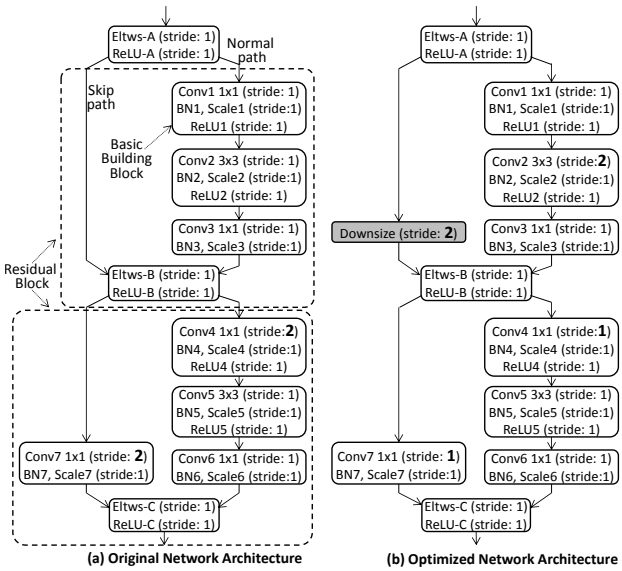


Fig. 1. ResNet with two residual blocks.

Conv7. Hence, stride value of ReLU-B is changed to 2 and the stride values of both Conv4 and Conv7 layers are changed to 1. Since ReLU-B doesn't have any dependency on neighboring pixels, the stride can be moved up to its parent layer. Hence, the stride value is changed to 2 in Eltws-B. To move the stride parameter further up beyond Eltws-B layer, all the prior layers in *normal* and *skip* paths needs to be carefully analysed to identify the data dependencies among the neighboring pixels and layers.

In *normal* path, the parent layers of ReLU-B starting from Scale3 to BN2 don't have any dependency on neighboring pixels. Hence, these layers can avoid processing of redundant pixels by changing their stride value to 2. The Conv2 layer (parent of BN2) can also avoid redundant data processing by changing the stride value to 2, because the subsequent layers don't use the generated redundant pixel data. However, the stride value of 2 can't be moved further up beyond Conv2 layer because the Conv2 layer uses a kernel size of 3x3 which makes the current pixel processing dependent on neighboring pixels which have to be generated by the prior layer ReLU1. Hence, the stride values are unchanged in ReLU1 and above layers. This completes the analysis of all the layers present in the *normal* path which results in moving the stride value 2 to Conv2 layer and using a stride value of 1 in the subsequent layers from BN2 till Scale3 as shown in Fig. 1(b). Hence, the feature map dimensions are reduced to half of the original dimensions for output of Conv2, and for both input and output of all the layers from BN2 to ReLU-B, resulting in 4x reduction in memory for these feature maps and 4x improvement in computations for all these layers.

In *skip* path, Eltws-B receives the input feature maps directly from ReLU-A having the unchanged original dimensions, whereas the input feature map dimensions from the *normal* path have been reduced to half of the original dimensions. As the dimensions of the feature maps from *skip* path

has to match with *normal* path to maintain the original functionality, the dimensionality mismatch needs to be handled to produce the same dimensions.

4. HANDLING DIMENSIONALITY MISMATCH

The feature map dimensions mismatch in optimized network needs to be handled to produce the same outputs as original network. We present three different approaches to handle the dimensionality mismatch in different scenarios.

4.1. Handle in destination layer

The feature map dimensionality mismatch can be directly handled in the destination layer Eltws-B itself which requires a modification in the inference framework. The Eltws layer implementation in inference framework needs to handle two different input dimensions with appropriate stride values. The lower dimension (*normal*-path) input stride value remains 1, where as the higher dimension (*skip*-path) input stride value is derived based on the *skip*-path to *normal*-path feature map dimensions ratio. For ResNet-50, since the ratio of skip-path to normal-path feature map dimensions is 2, the stride for the skip-path feature map is set to 2.

4.2. Downsize layer in skip-path

Approach in Section 4.1 requires a semantic change to include the stride parameter in Eltws layer. In cases where Eltws semantics change is not possible, dimensionality mismatch can be handled by adding a new layer '*downsize*' in the skip-path with a stride value derived based on the skip-path to normal-path feature map dimensions ratio. For ResNet-50, the stride value comes to be 2 for the downsize layer as shown in Fig. 1(b). The downsize layer is a copy function with one-to-one connection between input and output feature maps. It copies the necessary input feature map data to output buffer based on stride value. This approach requires a modification to the inference framework to support the downsize functionality.

4.3. Convolution layer in skip-path

If the inference framework modification is not possible, the dimensionality mismatch can be handled by using a convolution layer instead of a downsize layer in the skip-path with a stride value derived based on the skip-path to normal-path feature map dimensions ratio. The added convolution layer utilizes a kernel size of 1x1 with the weight values set to one. It will have a one-to-one connection between input and output feature maps. The added 1x1 convolution layer requires an extra multiplication for each output feature map pixel when compared to downsize.

Out of these three approaches, the first approach requires Eltws semantics change where as other two approaches require a layer to be added in the skip-path. The first two approaches require inference framework modification, but not required in the third approach. The Eltws semantics change approach is optimal compared to other two approaches as it doesn't require any extra layer processing. If semantics change is not possible, addition of a downsize layer is less complex when compared to a convolution layer in the skip-path as it doesn't require extra multiplication operations. However, these operations are negligible when compared to 4x reduction of operations achieved in the normal-path. Hence, a convolution layer in the skip-path is more appropriate to handle the dimensionality mismatch as it doesn't require any modifications to inference framework with negligible complexity.

5. AUTOMATIC OPTIMAL NETWORK GENERATION

The proposed method is realized using an automatic optimal network generation process. The design flow of our optimal network generation is shown in Fig. 2(a). A network graph is derived, where each node is a layer and each edge is a connection between two layers. We iterate through each node of the network graph based on their descending order of depth from the root node of the graph.

The current node pass through the condition "*Is All Child Stride Movable*" to check whether its children have a movable stride. This conditional check is satisfied when its children of the type convolution 1x1 layer with the stride value > 1. The movable stride "X" is an amount of children stride which can be moved to prior layer. This is derived from the Greatest Common Divisor (GCD) of its children strides. For an example, if the children strides are 4 and 6, the possible movable stride of these two children is 2. When the conditional check is satisfied, the stride X is moved to current node through "*Move_stride_X_to_current_node*" function and the strides of children are updated with a division by X factor through "*Divide_all_ child_stride_by_X*" function.

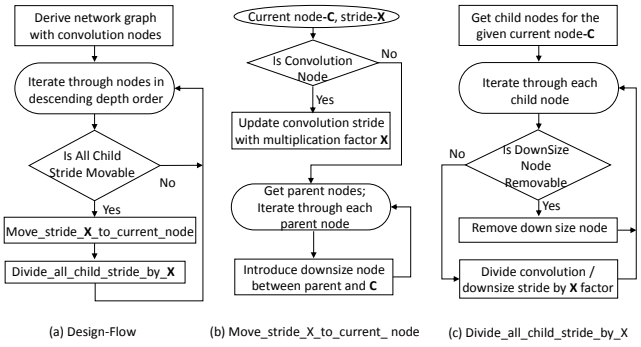


Fig. 2. Optimal network generation.

The *Move_stride_X_to_current_node* function flow diagram is shown in Fig. 2(b). If the current node "C" itself is a convolution layer, the stride is updated with the multiplication factor X. Otherwise, a downsize layer node is introduced with a stride value X between current node-C and its parent for the dimensionality reduction as the node-C layer may not have the stride parameter as per the semantics.

The downsize layer will have one-to-one connection between input and output feature maps. Its functionality is similar to convolution 1x1 layer with one-to-one connections among input and output feature maps (Section 4.3). Hence, the "*Is All Child Stride Movable*" condition checks whether its children of type not only convolution 1x1 layer, but also downsize layer with stride > 1. The introduced downsize layer is a temporary layer which may be removed over subsequent iterations of nodes in descending depth order in the process, through *Divide_all_child_stride_by_X* function.

The *Divide_all_child_stride_by_X* function flow diagram is shown in Fig. 2(c). It iterates through all child nodes of the given current node and checks whether the child node is a downsize layer with the stride equal to X. If it is true, the downsize layer node will be removed as it is unnecessary. Otherwise, the stride of the convolution or downsize layer is divided by X. If it is not removed due to the unsatisfied stride movable condition, it will stay in the final optimal graph.

Our optimal generation algorithm can support all three dimensionality mismatch handling approaches. To handle the dimensionality mismatch in destination layer such as Eltws layer, we simply remove the downsize layers from the final optimal graph generation as it is taken care by the updated Eltws layer in inference framework itself. To handle the dimensionality mismatch with downsize layer, the optimal graph generation has to do nothing as it already considered downsize layers. To handle the dimensionality mismatch with 1x1 convolution layer, we simply replace all downsize layers in the final optimal graph with 1x1 convolution layers with one-to-one connections.

The complexity of our optimal network generation algorithm is $O(M*(N+Q))$, where M is a total number of nodes in network graph, N is a maximum of number of children of all nodes in the graph, and Q is a maximum of number of parents of all nodes in the graph. In the worst case scenario, it is $O(M^2)$. Hence, it is negligible overhead compared to the convolution, BN, ReLU, etc. layers computation.

6. EXPERIMENTAL RESULTS

The proposed method is integrated with the Caffe framework and it is experimented on ResNet-50 inference. The CPU mode of Caffe is used to run the inference framework on a high-end Samsung smartphone with an octa-core ARM Cortex-A53 processor. The convolution kernels are hand-optimized with Neon SIMD for executing both original and optimized networks. The proposed method results in 4x layerwise gain in terms of number of operations and memory requirements for the layers in the residual block of Fig. 1(b) as shown in Table 1.

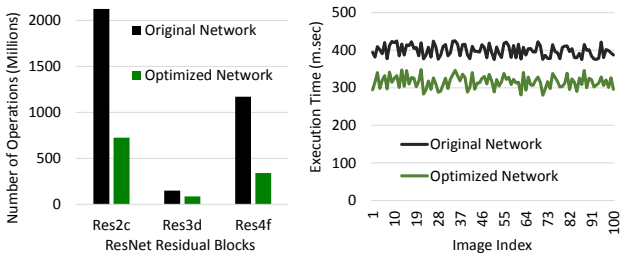
The proposed method has optimized 3 residual blocks in ResNet-50 network and reduced the number of operations and memory accesses by 1.7x to 3.4x for those residual blocks as shown in Fig. 3(a). We have generated the optimal networks

Parameter	Original Network	Optimized Network	Gain	Layers Gained
Memory Requirement per layer	Width x Height	(Width x Height)/4	4x	Conv2[1], BN2, Scale2, ReLU2, Conv3, BN3, Scale3 Eltws-B, ReLU-B
Processing Requirement per layer	Width x Height	(Width x Height)/4	4x	

[1] for Conv2, only output feature map memory is reduced by 4x

Table 1. Layerwise result obtained with optimal network.

using the dimensionality mismatch handling approaches presented in Section 4, and the performance is similar in all three cases with a variation of 0.1%. Hence, consideration of the convolution layer instead of a downsize layer in skip-path for dimensionality mismatch handling is more appropriate to generate an optimal network as it doesn't require any update in the framework.



(a) Residual Block level results (one image) (b) Execution time for 100 images

Fig. 3. Comparison of original and optimal networks.

The performance of the original and optimized networks are evaluated for image classification task on the ImageNet dataset [13]. The execution times of original and optimized networks of ResNet-50 for 100 images are shown in Fig. 3(b). We have achieved an average performance gain of $\sim$22% compared to the original network. The little performance variation among the images for both original and optimized networks are observed, which may be due to data cache effects. The total memory and performance gains acheived with our proposed method are shown in Table 2.

Parameter	Original Network	Optimized Network	Gain
Total Memory Requirement	~226 MB	~213 MB	~6 %
Performance on Samsung Smartphone with Octa-Core ARM Cortex-A53 Processor	~400 ms	~312 ms	~22%

Table 2. Memory and performance results.

The proposed method can be used as an on-line or off-line process. The method can be utilized for other networks if this redundancy pattern is present.

7. CONCLUSION

In this paper, we present a novel method to generate an optimal network by exploiting the redundancy patterns in the network. We have obtained $\sim$22% performance gain with optimized ResNet-50 on Samsung smartphone with an octa-core ARM Cortex-A53 processor. The proposed method augments the state-of-the-art research on pruning and compression techniques to enable the complex deep learning applications on embedded platforms. The promising results motivate the exploration of more redundancy patterns in CNNs to reduce the complexity and is our future work.

8. REFERENCES

[1] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, New York, NY, USA, 2014, MM '14, pp. 675–678, ACM.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1–9.

[4] J. Pang, H. Lin, L. Su, C. Zhang, W. Zhang, L. Duan, Q. Huang, and B. Yin, "Accelerate convolutional neural networks for binary classification via cascading cost-sensitive feature," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 1037–1041.

[5] J. Pasquet, M. Chaumont, G. Subsol, and M. Derras, "Speeding-up a convolutional neural network by connecting an svm network," in *2016 IEEE International Conference on Image Processing (ICIP)*, Sept 2016, pp. 2286–2290.

[6] Z. Yang, M. Moczulski, M. Denil, N. d. Freitas, A. Smola, L. Song, and Z. Wang, "Deep fried convnets," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1476–1483.

[7] Andrew Lavin and Scott Gray, "Fast algorithms for convolutional neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[8] Song Han, Huizi Mao, and William J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[9] Yiwen Guo, Anbang Yao, and Yurong Chen, "Dynamic network surgery for efficient dnns," *CoRR*, vol. abs/1608.04493, 2016.

[10] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng, "Quantized convolutional neural networks for mobile devices," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[11] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[12] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf, "Pruning filters for efficient convnets," *CoRR*, vol. abs/1608.08710, 2016.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.