LeanResNet: A Low-cost Yet Effective Convolutional Residual Networks

Jonathan Ephrath [1] **Lars Ruthotto** [2] **Eldad Haber** [3] [4] **Eran Treister** [1]

Abstract

Convolutional Neural Networks (CNNs) filter the input data using spatial convolution operators with compact stencils. Commonly, the convolution operators couple features from all channels, which leads to immense computational cost in the training of and prediction with CNNs. To improve the efficiency of CNNs, we introduce lean convolution operators that reduce the number of parameters and computational complexity, and can be used in a wide range of existing CNNs. Here, we exemplify their use in residual networks (ResNets), which have been very reliable for a few years now and analyzed intensively. In our experiments on three image classification problems, the proposed LeanResNet yields results that are comparable to other recently proposed reduced architectures using similar number of parameters.

1. Introduction

Convolution Neural Networks (CNNs) (LeCun et al., 1990) are among the most effective machine learning approaches for processing high-dimensional data and are indispensable in, e.g., in recognition tasks involving speech (Raina et al., 2009) and image (Krizhevsky et al., 2012) data.

In a CNN, the features are grouped into channels. Through the convolution operators, each feature interacts with other features from a small neighborhood in the same channel and, in most existing approaches, the features from the same neighborhood in the remaining channels; (Gu et al., 2018; Goodfellow et al., 2016). A drawback of this fully coupled approach is that the number of convolution operators in a layer is proportional to the product of the number of input and output channels. This scaling can be expensive when using wide architectures and leads to a large number of weights, often in the millions and beyond. It also complicates the deployment of such CNNs, especially on devices with limited memory resources.

In recent years there has been an effort to reduce the number of parameters in CNNs. Among the different approaches are the methods of pruning (Hassibi & Stork, 1992; Han et al., 2015; Li et al., 2017; Luo et al., 2017) and sparsity (Wen et al., 2016; Changpinyo et al., 2017; Han et al., 2016) that have been typically applied to reduce weights in full networks. It has been shown that once a network is trained, a large portion of its weights can be removed without hampering its efficiency by much. However, the non-zero structure of the weights in the resulting networks is typically unstructured, which may lead to inefficient deployment of the networks on hardware. Still, the success of pruning suggests that there is a significant redundancy in standard CNNs (Molchanov et al., 2016).

Another recent effort to reduce the number of parameters in networks is to define architectures based on "depth-wise" separable convolutions, which are block diagonal convolution operators. The depth-wise convolution restricts the interaction of each feature to its nearby features in the same channel. To facilitate coupling across the channels, the depth-wise operators is typically used in conjunction with point-wise 1×1 convolutions. This was applied in the works of (Howard et al., 2017; Sandler et al., 2018; Wang et al., 2016; Zhang et al., 2018; Ma et al., 2018), together with either bottleneck or shuffling techniques. Since these works use the depth-wise and 1×1 separately, with activation and batch normalization layers in between them, they require a redesign of existing CNN architectures. In addition, applying the depth-wise convolution on its own has a high ratio of floating point operations (FLOPs) to memory access, which has led to the design of networks with shifts instead of convolutions (Wu et al., 2018). It is known, however, that memory access is the true bottleneck in modern parallel hardware, and not necessarily FLOPs. In fact, the work (Qin et al., 2018) suggests a superior implementation of the depthwise convolution on GPUs, which is involved with more FLOPs than necessary.

In this paper, we propose a novel way to parameterize CNNs more efficiently, while simply keeping the same structure of the known networks, e.g. residual networks (ResNets) (He et al., 2016a;b), which have been one of the

[1]Department of Computer Science, Ben Gurion University, Be'er Sheva, Israel. [2]Departments of Mathematics and Computer Science, Emory University, Atlanta, GA, USA. [3]Department of Earth, Ocean and Atmospheric Sciences, University of British Columbia, Vancouver, Canada. [4]Xtract AI, Vancouver, Canada. Corresponding author: Eran Treister: erant@cs.bgu.ac.il.

most reliable architectures in the literature. Our goal is to reduce the number of weights in the networks and the costs of training and evaluating the CNN. Similarly to recent approaches we use depth-wise convolutions, and 1×1 convolutions to impose coupling between channels. The following three aspects set our work apart from other approaches: (1) We *linearly* add the depth-wise and 1×1 convolutions so that the two operations can be applied simultaneously in hardware, in the same memory read. (2) The combined convolution can be simply used as a single convolution operator instead of the standard convolution in any existing CNN, without any structural changes to the architecture. (3) We use a 4-point stencil only instead of the standard 3×3 or larger stencil, to further reduce memory access and FLOPs of the depth-wise convolution.

2. ResNets with Lean Convolution Operators

We consider a standard residual network (ResNet) (He et al., 2016a;b) as a baseline architecture, since it has been very successful and reliable for many tasks. Given a data sample $\mathbf{y}_0$, the forward propagation through the network is defined by a series of steps, where the jth step is given by

$$\mathbf{y}_{j+1} = \mathbf{y}_j + \mathbf{F}(\boldsymbol{\theta}_j, \mathbf{y}_j), \quad \text{for} \quad j = 0, \dots, N-1. \quad (1)$$

Here, $\boldsymbol{\theta}_j$ is the set of weights associated with the jth step. The nonlinear term in (1) usually reads

$$\mathbf{F}(\boldsymbol{\theta}, \mathbf{y}) = \mathbf{K}_2(\boldsymbol{\theta}^{(2)})\sigma(\mathcal{N}(\mathbf{K}_1(\boldsymbol{\theta}^{(1)})\sigma(\mathcal{N}(\mathbf{y})))), \quad (2)$$

where $\sigma(x) = \max\{x, 0\}$ denotes a element-wise rectified linear unit (ReLU) activation function, the weights are divided into $\boldsymbol{\theta}^{(1)}$ and $\boldsymbol{\theta}^{(2)}$ that parameterize the two linear operators $\mathbf{K}(\boldsymbol{\theta}^{(1)})$ and $\mathbf{K}(\boldsymbol{\theta}^{(2)})$. $\mathcal{N}$ denotes a normalization layer that has trainable parameters as well (omitted here for ease of presentation). The operators $\mathbf{K}_1$ and $\mathbf{K}_2$ are composed of spatial convolution operators. If the input $\mathbf{y}$ has c_{in} channels, and the output $\mathbf{K}_1\mathbf{y}$ has c_{out} channels, then a common choice for $\mathbf{K}_1$ is a $c_{\text{out}} \times c_{\text{in}}$ block matrix of convolutions, introducing full coupling across the channels.

Our lean convolution operator contains two types of operators. One is the depth-wise (block diagonal) operator which operates on each channel separately, and the other is a 1×1 convolution. For example, if $c_{\text{in}} = c_{\text{out}} = 4$, then in matrix form, the operator is given by

$$\mathbf{K}_{\text{lean}} = \begin{pmatrix} \hat{\mathbf{C}}_1 & \alpha_{1,2}\mathbf{I} & \alpha_{1,3}\mathbf{I} & \alpha_{1,4}\mathbf{I} \\ \alpha_{2,1}\mathbf{I} & \hat{\mathbf{C}}_2 & \alpha_{2,3}\mathbf{I} & \alpha_{2,4}\mathbf{I} \\ \alpha_{3,1}\mathbf{I} & \alpha_{3,2}\mathbf{I} & \hat{\mathbf{C}}_3 & \alpha_{3,4}\mathbf{I} \\ \alpha_{4,1}\mathbf{I} & \alpha_{4,2}\mathbf{I} & \alpha_{4,3}\mathbf{I} & \hat{\mathbf{C}}_4 \end{pmatrix}, \quad (3)$$

where $\alpha_{i,j}\mathbf{I}$ is a scaled identity defined by a learned scalar parameter $\alpha_{i,j}$. The operator $\hat{\mathbf{C}}_i$ is a matrix that corre-

sponds to a 5-point convolution kernel

$$\begin{bmatrix} 0 & c_{i,1} & 0 \\ c_{i,2} & \alpha_{i,i} & c_{i,3} \\ 0 & c_{i,4} & 0 \end{bmatrix}, \quad (4)$$

where $\alpha_{i,i}$ is the i, i entry of the 1×1 convolution, and $c_{i,1}, ..., c_{i,1}$ are additional 4 parameters per input channel i. $\mathbf{K}_{\text{lean}}$ has $c_{in} \times c_{\text{out}} + 4c_{in}$ parameters can be used instead of the standard operators in CNNs. We note that if the number of input channels is larger than 4, then the 1×1 convolution is the dominating operator both in terms of parameters and FLOPs.

Interpretation ResNets have been recently interpreted as time-dependent nonlinear PDEs (Haber & Ruthotto, 2017; Chang et al., 2018; Weinan, 2017; Chaudhari et al., 2017; Lu et al., 2018; Ruthotto & Haber, 2018; Chen et al., 2018), which allows the community to analyze and extend ResNets using theoretical and practical ideas from the world of ODEs and PDEs. In this point of view, the depth-wise 3×3 convolution can be seen as a linear combination of a mass term, and discretization of first and second spatial derivatives in each dimension. The 1×1 convolution approximates a mass term only. It is known that most simple spatial derivatives can be approximated by a five-point stencil as in (4), and therefore, a 3×3 stencil may be unnecessary for extracting features in CNNs. A tremendous advantage will be made in 3D CNNs where the standard 27-point convolutions are replaced with a 7-point stencil (the 3D version of (4)).

3. Experiments

We experimentally compare the architectures proposed in this paper to a ResNet with fully-coupled convolutions, and other reduced architectures: ShuffleNetV2 (Ma et al., 2018), MobileNetV2 (Sandler et al., 2018), and ShiftResNet (Wu et al., 2018). We use the CIFAR-10, CIFAR100 (Krizhevsky & Hinton, 2009) and STL-10 (Coates et al., 2011) data sets. Our primary focus is to compare how the different architectures perform using a relatively small number of weights. Our experiments are performed with the PyTorch software (Paszke et al., 2017).

We adopt a rather standard ResNet architecture, and demonstrate the performance of its lean version. Our ResNet networks consist of several blocks, that are preceded by an opening convolutional 3×3 layer, that initially increases the number of channels. Then, there are several blocks, each consisting of a ResNet based part with a number of steps that varies between the different experiments. Each convolution is applied in addition to a ReLU activation and batch normalization as described in (1). The last block consists of a pooling layer that averages each channel's map to

	CIFAR10			CIFAR100			STL10		
Architecture	Network	Params	Val. acc.	Network	Params	Val. acc.	Network	Params	Val. acc.
ResNet	A	4.3M	94.7%	C	27M	78.5%	E	17M	84.1%
ResNet (small)	B	0.6M	91.6%	D	3.8M	72.3%	F	1.8M	78.6%
MobileNetV2	A	0.5M	91.5%	C	2.7M	71.6%	E	1.9M	80.4%
ShuffleNetV2	0.5x	0.4M	89.3%	1.5x	2.5M	70.6%	1.5x	2.5M	**86.2%**
ShiftResNet	A	0.5M	92.5%	C	3.1M	74.2%	E	1.9M	82.3%
LeanResNet [ours]	A	0.5M	**92.8%**	C	2.9M	**74.3%**	E	2.0M	83.7%

Table 1. Classification results

a single pixel, and we use a fully-connected linear classifier with softmax and cross entropy loss. In Table 2 we summarize the network parameters that we use, which differ in the number of channels and the number of repetitions for each layer.

Type	Layer width	# Steps
A	32-64-128-256	2-3-3-3
B	12-24-48-96	2-3-3-3
C	64-128-256-512	3-5-7-4
D	24-48-96-192	3-5-7-4
E	32-64-128-256-512	2-3-3-3-3
F	12-24-48-96-192	2-3-3-3-3

Table 2. Network configurations

As noted, although the architectures of LeanResNet and ResNet appear to be the same, LeanResNet is based on the parameterized convolution (3), hence it consumes less parameters. The convolution sizes of MobileNetV2 and ShiftResNet were chosen such that the size of the expanded (by 6) 1×1 convolution in a layer is equivalent to the size of a square 1×1 convolution of LeanResNet. The architecture of ShuffleNetV2 is evaluated with the configurations (0.5x,1.0x,1.5x) that were introduced in the papers. For training the networks we use the ADAM optimizer (Kingma & Ba, 2014) and a minibatch of 100. We run 300 epochs and reduce the learning rate by a factor of 0.5 every 75 epochs, starting from 0.1. We also used standard data augmentation, i.e., random resizing, cropping and horizontal flipping.

Our classification results are given in Table 1. The results show that our architecture is in par and in some cases better than other networks. There is no preferred architecture between all options, but our architecture has the advantage of simplicity and resemblance to a standard and reliable ResNet network. We note that although not shown here for a fair comparison, it is better to use 3×3 convolutions in the early layers (where there are low numbers of channels and parameters), and then switch to reduced architectures as the network progresses and the number of channels grow.

3.1. Computational Performance

We compare the computational cost of our CUDA implementation of the lean convolution with two other combi-

nation of layers, comprised of a 1×1 convolution that is followed by a depth-wise convolution. In one combination we use $c_{in} = c_{out}$, and in the other $c_{in} \approx 6c_{out}$, but with the same number of parameters. Such layers are applied in (Sandler et al., 2018). We compare the runtime of a typical network: the first layer consists of 16 channels of 512×512 maps, and the maps are coarsened by a factor of 2 when the channels increase by a factor of 2 (i.e., for 512 channels the images are of size 16). We use a batch size of 64, and compare the runtime of a NVIDIA GeForce 1080Ti GPU for the task. The implementation for the other convolutions is based on PyTorch's 1×1 and grouped convolutions using CUDA 9.2. Figure 1 summarizes the results. The depth-wise convolutions dominate the low channels layers, while all combination converge to the cost of the 1×1 convolution as the channels increase (and the depthwise layer becomes negligible). Our implementation of (3) is clearly faster and exploits the simultaneous multiplication of the operators.
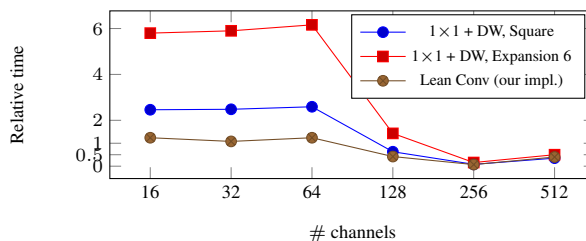

Figure 1. Relative timings of reduced convolutions compared to a 3×3 convolution (lower is faster). The expanded and square 1×1 convolutions has the same number of parameters.

4. Conclusion

We present a lean convolution operator that aims at reducing the number of parameters and computational costs of CNNs. In our experiments the new architecture yields classification results that are comparable to other reduced architectures, and is almost as effective as a fully-coupled ResNet. It is important to realize that our new architecture becomes even more advantageous for 3D or 4D problems, e.g., when analyzing time series of medical or geophysical images, the cost of each convolution is much more expensive. Also, the number of weights in the 3D kernels imposes memory-related challenges.

5. Acknowledgements

References

Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., and Holtham, E. Reversible architectures for arbitrarily deep residual neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Changpinyo, S., Sandler, M., and Zhmoginov, A. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*, 2017.

Chaudhari, P., Oberman, A., Osher, S., Soatto, S., and Carlier, G. Deep Relaxation: Partial Differential Equations for Optimizing Deep Neural Networks. pp. 1–22, 2017.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pp. 6571–6583, 2018.

Coates, A., Ng, A., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the 14th AISTATS*, pp. 215–223, 2011.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, November 2016.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., and Chen, T. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, May 2018.

Haber, E. and Ruthotto, L. Stable architectures for deep neural networks. *Inverse Problems*, 34(1), 2017.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efcient neural network. *International Journal of Computer Vision*, 5(5):1135–1143, 2015.

Han, S., Pool, J., Narang, S., Mao, H., Tang, S., Elsen, E., Catanzaro, B., Tran, J., and Dally, W. J. Dsd: regularizing deep neural networks with dense-sparse-dense training flow. *arXiv preprint arXiv:1607.04381*, 3(6), 2016.

Hassibi, B. and Stork, D. G. Second order derivatives for network pruning: Optimal brain surgeon reconstruction. *International Journal of Computer Vision*, 5(5):164–171, 1992.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pp. 630–645. Springer, 2016b.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. Imagenet classification with deep convolutional neural networks. *Adv Neural Inf Process Syst*, 61:10971105, 2012.

LeCun, Y., Boser, B. E., and Denker, J. S. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pp. 396–404, 1990.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient ConvNets. https://arxiv.org/abs/1608.08710, 2017.

Lu, Y., Zhong, A., Li, Q., and Dong, B. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning (ICML)*, 2018.

Luo, J.-H., Wu, J., and Lin, W. ThiNet: A filter level pruning method for deep neural network compression. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 116–131, 2018.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*, 3, 2016.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems*, 2017.

Qin, Z., Zhang, Z., Li, D., Zhang, Y., and Peng, Y. Diagonalwise refactorization: An efficient training method for depthwise convolutions. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2018.

Raina, R., Madhavan, A., and Ng, A. Y. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pp. 873–880. ACM, 2009.

Ruthotto, L. and Haber, E. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*, 2018.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Wang, M., Liu, B., and Foroosh, H. Design of efficient convolutional layers using single intra-channel convolution, topological subdivisioning and spatial" bottleneck" structure. *arXiv preprint arXiv:1608.04337*, 2016.

Weinan, E. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11, March 2017.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

Wu, B., Wan, A., Yue, X., Jin, P., Zhao, S., Golmant, N., Gholaminejad, A., Gonzalez, J., and Keutzer, K. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9127–9135, 2018.

Zhang, X., Zhou, X., Lin, M., and Sun, J. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, 2018.