Structural Compression of ResNet-like Convolutional Neural Networks using Separable Convolutions

Nicholas Kastanos (nk569), Queens' College
Department of Computer Science and Technology
University of Cambridge
Cambridge, CB3 0FD
`nk569@cam.ac.uk`

Abstract

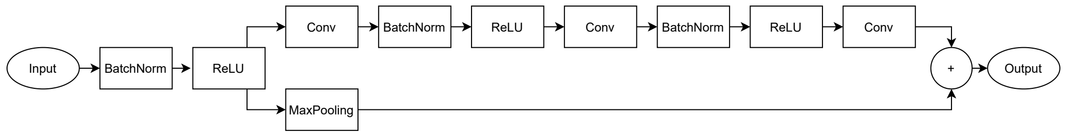
Convolutional neural networks (CNNs) are commonly used in computer vision neural networks, however inference on very deep models is resource intensive. In order to apply these CNNs to resource constrained devices, models must be compressed. By replacing convolution layers with separable convolutions, the number of parameters and multiply-accumulate operations can be reduced at the cost of performance. In this report, the compressibility of ResNet50V2 CNN model is investigated by replacing convolutions with spatial and depthwise convolutions. Additionally, the effects of 8-bit quantized compression on these layers is investigated. It is shown that a 8-bit compressed ResNet50V2 model with depthwise convolutions has a compression factor of 0.15 while only experiencing a 1.34 % decrease in accuracy.

1 Introduction

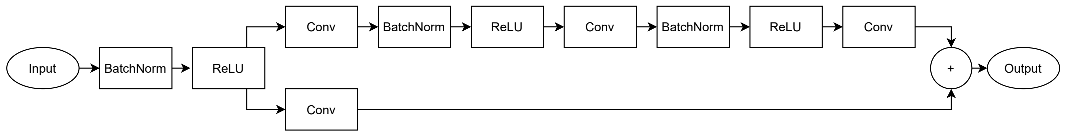
Much of the research into convolutional neural networks (CNNs) is focused around increasing the performance of architectures for computer vision tasks, most commonly image classification and object detection [1, 2, 3, 4, 5]. The applications of these neural networks, however, require efficient distributed training, fast inference times, and deployment to resource constrained hardware [6, 7].

Newer networks such as MobileNet [7], ShuffleNet [8], and EffNet [9] each use different forms of separable convolutions to reduce the number of parameters and multiply-accumulate operations (MACs), while maintaining model performance. This structural compression allows CNNs to be applied to previously unavailable applications, such as autonomous vehicles, cellphones, and robotics. While the benefits of separable convolutions can be seen in the networks which implement them, many other structural changes are also introduced. Therefore, the precise impact of separable convolutions ambiguous.

In this paper, the effects of two types of separable convolutions – spatial and depthwise – are investigated in the ResNet50V2 architecture [10]. This is achieved by replacing convolutional layers with separable variants, while making no other structural changes. Additionally, the effects of 8-bit quantization on the separable convolutions are investigated. Section 2 gives contextual information on the CNN architecture and optimizations. The experiments use the TensorFlow framework [11] for training and evaluation. Section 3 details the experiment setup, while Section 4 contains the experiment results and discussion. The source code used for the development of the neural networks can be accessed at `https://github.com/Nicholas-Kastanos/tf-cnn-compress`.



(a) Identity passthrough



(b) Bottleneck passthrough

Figure 1: Residual blocks used in ResNetV2

2 Related Work

2.1 Residual networks

The residual block first postulated for use in ResNet has become a common-place feature in many subsequent networks [1, 4, 12]. It was designed to combat network degradation in very deep CNNs, where more complicated features were being learned than were required. The skip connections introduced by the residual block allow features to bypass convolutional blocks and be passed on to subsequent layers unaltered. This can be summarised in the formula $g(x) = f(x) + x$ where x is the input, $g(x)$ is the desired output, and $f(x)$ is the input affected by the convolutional block.

This architecture was further developed until the form presented in ResNetV2 [10], which includes two variants: identity passthrough which applies pooling to the skip connection x; and bottleneck passthrough, which includes a single convolution in the skip connection. These two residual blocks can be seen in Figure 1.

2.2 Separable convolutions

Convolution layers contain a vast majority of the parameters in modern CNNs [4, 7, 10]. By targeting parameter reductions to these layers, the compression can be spread throughout the network. Separable convolutions reduce the number of parameters by decomposing the convolution into multiple stages through spatially and depthwise separable convolutions. While these convolutions reduce the memory and computation requirements of the system, the reduced parameters limits the number of possible kernels explored in training, and the resulting network may be suboptimal.

2.2.1 Spatially separable convolutions

A convolution kernel can be decomposed on its 2D spatial axis, i.e. height and width [4, 5, 9, 13]. Conceptually, a $n \times n$ kernel can be separated into two smaller kernels, a $n \times 1$ followed by a $1 \times n$ kernel. These kernels can be applied in sequential convolutions to obtain the same output shape as the single convolution. These decomposed kernels scale the parameters required by the convolution by a factor $P_s(n)$ (see Equation 1). Similarly, the MAC operations of a spatially separated convolution are reduced. For a $N \times N$ input convolved with a $n \times n$ kernel, the number of multiplications are reduced by a factor of $M_s(n)$ (see Equation 2).

Equations 1) and 2 show that spatially separable convolutions show computational benefits when $n > 2$.

$$P_s(n) = \frac{2}{n} \tag{1}$$

$$\begin{aligned} M_s(N, n) &= \frac{2}{n} + \frac{2}{n(N-2)} \\ \Rightarrow M_s(n) &= \frac{2}{n}, \text{ where } N >> n \end{aligned} \tag{2}$$

2.2.2 Depthwise separable convolutions

Depthwise separable convolutions separate the spatial convolution from the depth of the filters [7, 14]. This is accomplished by an initial depthwise convolution, followed by a pointwise convolution. The initial depthwise convolution separates the channels of the input and kernel, and convolves them independently. The pointwise convolution is a $N_F \times 1 \times 1 \times N_C$ convolution where N_F and N_C are the number of filters and channels respectively.

The number of parameters $P_d(n)$ and multiplications $M_d(n)$ are reduced by the same factor, which can be seen in Equation 3. Many CNNs have $N_F >> 1$, therefore depthwise convolutions show compression kernel sizes greater than 1 [2, 3, 10].

$$P_d(n) = M_d(n) = \frac{1}{n^2} + \frac{1}{N_F} \approx \frac{1}{n^2} \tag{3}$$

2.3 Quantization and datatype compression

Many resource-constrained devices do not have sufficient memory to use large neural networks or may not have access to floating-point arithmetic units. Both factors can be mitigated by using low-precision integer datatypes, such as 8-bit integers [6, 15]. This effectively reduces the memory required for each parameter by 1/4. However, by reducing the precision of the datatype, the learned parameters are not represented fully, reducing the performance of the network. This has been shown to have significant effect on the network when reducing 32-bit floating point numbers to 8-bit integers, however quantization aware training can limit the impact [6].

TensorFlow, by default, uses 32-bit floating point precision for its network layers and training [11]. TensorFlow Lite provides functionality to convert trained models to quantized, 8-bit integer models, however it does not have native 8-bit implementations for all layer types. In the event an incompatible layer is used, the engine upscales the activations to the full precision for that layer.

3 Methodology

In order to asses the benefits and costs of the compression, a baseline architecture is established and used as a reference point. The ResNet50V2 image classification CNN-based neural network is used as the baseline architecture. The baseline architecture and subsequent networks are trained using the same parameters.

3.1 Training

The CNNs are trained using the CIFAR-10 dataset [16] in batches of 32, with random horizontal flips, rotations, and cropping data augmentations. The Adam optimizer [17] is used with a learning rate of 0.001, and categorical crossentropy loss. The CIFAR-10 training split is separated into train and validation splits. The splits are not shuffled to ensure each model is trained on the same train and validation datasets, and the validation split contains 10% of the total training data. The dataset's test split is used to evaluate the models on new data. The networks are trained until no improvement in the validation loss has been observed for 10 epochs, at which point the best results weights are saved. This ensures that each model has enough time to learn its optimal weights.

3.2 Modified residual blocks

In order to modify the ResNetV2 network, the convolutions in the residual blocks are chosen to be substituted with separable convolutions. Additional batch normalization and activation layers are not added between these new layers. An example of this expansion can be seen in Figure 2. Only convolutions which would provide parameter and compute reduction are replaced with separable convolutions. For ResNet50V2, these are any convolutions where the kernel size $n > 1$.

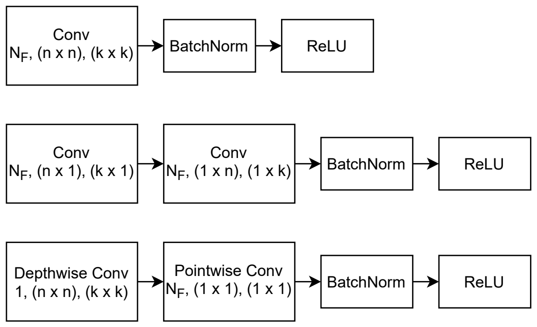


Figure 2: Example of replaced convolutions with separable convolutions with N_F filters, kernel size n, and stride k. From top to bottom: Traditional Convolution; Spatially Separable Convolution; Depthwise Separable Convolution

3.3 Post-training quantization and compression

The fully trained models are compressed to 8-bit precision using TensorFlow Lite. The framework recommends the provision of the representative dataset to enhance the optimization process. This allows for quantization of both weights and activations for quantizable operations. The representative dataset is compiled using 1000 samples from the training dataset.

4 Results and evaluation

The trained CNNs are evaluated using the test split of CIFAR-10. The dataset includes 1000 samples from each of the 10 classes, ensuring class balance. The models are evaluated by creating TensorFlow Lite models from the original model. In the case of compressed models, the TensorFlow Lite model converter optimizes the models and quantises the activations using the representative dataset.

Table 1 shows the total parameters in the ResNet50V2 models. This shows that substituting separable convolutions into a CNN, the total number of parameters can be reduced by up to $57\ \%$. Table 1 also shows training performance of the models.

Table 2 shows the results of the analysis. The compression factor is the ratio of the model size under investigation to that of the ResNet50V2 baseline model. As seen in Table 2, applying spatial convolutions to ResNet50V2 has a compression factor of 0.84 while the drop in accuracy is $0.54\ \%$. Depthwise convolutions result in an accuracy drop of $1.25\ \%$ while compressing the network with compression factor of 0.57.

Table 1: Training statistics

Convolution Type	Total Parameters	Best Epoch	Train Loss	Validation Loss
Full	$23,585,290$	54	0.2539	0.2560
Spatial	$19,812,874$	70	0.2478	0.2888
Depthwise	$13,559,498$	47	0.4447	0.3516

Table 2: Results using the test split of CIFAR-10

Convolution Type	Loss	Accuracy (%)	Size (kB)	Compression Factor
Full	1.0045	74.63	$94,145$	1
Spatial	1.0334	74.09	$79,064$	0.84
Depthwise	0.8940	73.38	$54,050$	0.57
Compressed Full	1.3332	73.99	$24,446$	0.26
Compressed Spatial	1.4457	73.61	$20,770$	0.22
Compressed Depthwise	1.0686	$\mathbf{73.29}$	$14,517$	$\mathbf{0.15}$

Applying both depthwise convolution substitution and 8-bit quantized compression results in a model approximately 6.5 times smaller than the full ResNet50V2 model. This model does not significantly impact the performance of the model as seen by the accuracy decreasing by only $1.34\ \%$. In addition to storage size, the maximum RAM requirements can be analysed by `tflite-tools` [18]. The uncompressed models require a maximum of $196.61\ kB$, while the compressed versions require $49.15\ kB$. This memory requirement is constant across convolution types because the most memory intensive layers of ResNet50V2 are input processing layers.

The significant reduction in parameters and memory improvements shown by separable convolutions with very small drawbacks allows complicated CNN networks to be implemented on resource constrained devices. This is especially true for larger CNNs such as ResNet152V2 or YOLO object detection networks, as they are too large to operate uncompressed on MCU or FPGA devices. Efficient computer vision CNNs can be used to operate self-driving vehicles, drones, and other robotics. This allows the applications to be smaller, and more energy efficient, allowing for more possible applications.

Large benefits are observed for ResNet50V2, however, these benefits are not without limits. The separable convolutions were only applied to 3×3 kernels, as ResNet does not include larger kernel sizes on a regular basis. While Equations 1, 2, and 3 show that the compressive benefits of separable convolutions increase with kernel size, so too does the degradation in performance. This performance loss is not quantified by the results seen in Table 2.

5 Future Work

In order to determine the validity of separable convolutions for larger networks, further investigation into larger kernel and input sizes is required. If these convolution decompositions do not result in significant performance degradations, further improvements to CNN compression can be observed.

The initial depthwise convolution is $n \times n$, which can be separated spatially. The combination of these two techniques can result in further optimisation.

6 Conclusion

ResNet-like neural networks are common in image applications, however their large size and complexity prevents them from being used in resource constrained scenarios. By replacing the convolution layers with spatially and depthwise separable convolutions, the memory and compute intensive convolutional layers can be compressed. These separable convolution layers were applied to the ResNet50V2 image classification CNN trained on the CIFAR-10 dataset, along with 8-bit quantization. The smallest resulting network has a compression factor of 0.15 while only observing a $1.34\ \%$ decrease in accuracy. The now compressed network can be deployed to resource constrained devices to classify images at high accuracy while maintaining a low computational overhead.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] A. Bochkovskiy, C.-Y. Wang, and H. Liao, "Yolov4: Optimal speed and accuracy of object detection," *ArXiv*, vol. abs/2004.10934, 2020.

[4] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[5] J. Wang, H. Xiong, H. Wang, and X. Nian, "ADSCNet: asymmetric depthwise separable convolution for semantic segmentation in real-time," *Applied Intelligence*, vol. 50, no. 4, pp. 1045–1056, 2020.

[6] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, (Red Hook, NY, USA), p. 5151–5159, Curran Associates Inc., 2018.

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

[8] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, 2018.

[9] I. Freeman, L. Roese-Koerner, and A. Kummert, "Effnet: An efficient structure for convolutional neural networks," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 6–10, IEEE, 2018.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*, pp. 630–645, Springer, 2016.

[11] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[12] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[13] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *ArXiv*, vol. abs/1602.07360, 2017.

[14] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

[15] T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, and M. Aleksic, "A quantization-friendly separable convolution for mobilenets," in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pp. 14–18, 2018.

[16] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

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

[18] E. Liberis, "tflite-tools." `https://github.com/eliberis/tflite-tools`.