Comparing Optimization Methods of Neural Networks for Real-time Inference

Mir Khan*, Henri Lunnikivi*, Heikki Huttunen, Jani Boutellier
University of Tampere
Tampere, Finland
mir.markhan@tuni.fi, henri.lunnikivi@tuni.fi, heikki.huttunen@tuni.fi, jani.boutellier@tuni.fi
*Indicates equal contribution

***Abstract*—This paper compares three different optimization approaches for accelerating the inference of convolutional neural networks (CNNs). We compare the techniques of separable convolution, weight pruning, and binarization. Each method is implemented and empirically compared in three aspects: preservation of accuracy, storage requirements, and achieved speed-up. Experiments are performed both on a desktop computer and on a mobile platform using a CNN model for vehicle type classification. Our experiments show that the largest speed-up is achieved by binarization, whereas pruning achieves the largest reduction in storage requirements. Both of these approaches largely preserve the accuracy of the original network.**

Keywords: convolutional neural networks, model optimization, image classification

1. Introduction

Convolutional neural networks (CNNs) have demonstrated notable performance in a range of practical tasks, including image classification [14][3], face recognition [15][17], audio classification [11], and speech recognition [5]. At the same time as CNNs have become significantly popular, their deployment to small-scale devices has been hindered by the considerable memory- and computation time requirements of CNN models. To address this issue, several approaches [10][8][4][6] for CNN *model optimization* have been developed in the recent years.

Some techniques such as *weight pruning* [6] aim to reduce the number of weights in the model, which in turn reduces memory requirements for running and storing the model. It has also been shown that a pruned network with sufficient sparsity allows the computations to be performed through efficient procedures for handling sparse matrices, such as sparse matrix-vector multiplication [2]. Studies (e.g. [6]) have shown that a considerable portion of network weights can be removed without significantly impacting accuracy: the authors of [6] report $13\times$ reduction of memory requirements with no loss in accuracy.

The *separable convolution* (SepConv) approach introduced in [10] approximates 2D convolutions with convolutions by vectors, i.e. separable rank-1 kernels. The separable kernels can be obtained either by training the network from scratch with separable filters [1] or by formulating the process as an optimization problem where the reconstruction error of the feature maps using separable filters is minimized. These approaches [10][1], report speedups between $2\times$ to $4\times$ on CPU implementations.

Neural network *binarization* is a CNN quantization technique that has been introduced in [8]. The principal idea of binarization is to quantize weights, activations and intermediate computations to values of $+1$ and -1. The authors of [8] show a speedup of up to $7\times$ on a network for classifiying the MNIST dataset. Later works [18] extended the binarization to larger datasets such as ImageNet.

The SepConv and weight pruning approaches are similar in the sense that both approximate and optimize the CNN model by *reducing* the number of parameters, while keeping the precision of the remaining weights the same as in the original model; in general as 32-bit floats. In contrast, the CNN binarization optimization keeps the *number* of weights the same as in the original model, but reduces the precision of each weight. Based on existing works, it is not clear how these two alternative approaches compare in terms of preserving model accuracy, improving computation time, and reducing memory space requirements.

This paper follows optimization approaches based on separable convolution [10], binarization [8][4] and weight pruning [6], and applies them to the practical use case of CNN-based vehicle type recognition [9]. In case of the binarization optimization, our results mostly build on our previous work [12]. The different optimization approaches are compared in terms of performance, storage requirements, amount of computations and the network's accuracy compared to the uncompressed baseline.

The contributions of this work are summarized as follows:

- Detailed empirical analysis and comparison of three different model optimization methods for compression and speeding up of neural networks, namely separable convolution, weight pruning, and binarization.
- Open source optimized implementations of each method for efficient real-time inference.

Our results show that pruning offers the highest reduction in storage requirements, while the lowest execution time is attained with a binarized convolutional neural network

978-9-0827-9703-9/19/$31.00

Figure 1. From left to right, a 'normal car', 'bus', 'truck', and a 'van'.

(BCNN). The source code for our implementations are publicly available [1][2].

2. Network Model

Our basic neural network model is that of the vehicle classifier network presented in [12], with two convolutional layers, each with 32 output feature maps, and filter dimensions 5×5. Each convolutional layer is followed by a 2×2 maxpool operation. The convolutional layers are followed by three fully connected layers with shapes 24×24×32×100, 100×100, and 100×4. Each optimization approach we are using slightly alters the structure of the network. This will be discussed in Section 3.

The dataset used for training the network consists of 6555 photos of vehicles captured by a camera and manually categorized into four categories: *bus*, *normalcar*, *truck*, and *van*. Each image is of size 96×96 and is in full color. The data has been split into a training set (80%), validation set (10%) and a test set (10%). We use the accuracy recorded on the test set that corresponds to the best validation set accuracy as our final accuracy report. Figure 1 shows an example image from each class.

We implement all of our training procedures, including the custom model optimization approaches, in TensorFlow. For inference performance measurements, we have our own implementations for each model optimization approach. The GPU implementations are done in CUDA or OpenCL, depending on the platform tested.

3. Optimization Methods

In this section, we discuss in detail in each subsection the different optimization approaches we use and how they are incorporated to our network model. Each subsection begins by introducing the algorithmic changes, and then proceeds to explain how the method is implemented for inference purposes.

3.1. Separable Convolution

Convolutional layers in a CNN can be replaced by spatially separable convolution layers to reduce the number of computations and the number of parameters [10]. Essentially, a separable convolution layer will approximate a standard convolutional layer as follows

$$\sum_{c=1}^{C} Z_c^f * x_c \approx \sum_{k=1}^{K} h_k^f * \sum_{c=1}^{C} v_c^k * x_c \tag{1}$$

In Equation 1, $*$ denotes the convolution operation. The convolution of the image $x \in \mathbb{R}^{W \times H \times C}$ with each $Z^f \in \mathbb{R}^{5 \times 5 \times C}$, which is full rank, is approximated by two convolutions with $h^f \in \mathbb{R}^{1 \times 5 \times K}$, and $v^k \in \mathbb{R}^{5 \times 1 \times C}$. C is the number of channels (usually three), and K denotes the number of intermediate feature maps, which are computed by convolving the image by each of v^k (K in total), which are then in turn convolved by each of h^f kernels (F in total, corresponding to the original number generated by the full rank kernel Z^f). The result in Equation 1 is the computation of the fth feature map. The choice of number of intermediate feature maps K has an impact on the number of computations, memory requirements, and the network's accuracy. Therefore, it is imperative to carefully select a value for K that offers the best trade off over all these factors.

In our application, we choose the smallest value for K that achieves the best accuracy. We discuss this in more detail in Section 4. The separable filters can either be obtained by training the network with separable filters [1] or by minimizing the reconstruction error of the outputs of each convolution layer [10], which allows us to state the loss to be minimized with respect to the separable filters as follows (for the fth set of filters in the lth layer):

$$\mathcal{L}(\mathbf{X}) = \frac{1}{N} \sum_{x \in \mathbf{X}} \Big\| \sum_{c=1}^{C} Z_c^f * \Psi_{l-1}(x) - \sum_{k=1}^{K} h_k^f * \sum_{c=1}^{C} v_c^k * \Psi_{l-1}(x) \Big\|_2^2, \tag{2}$$

where the output of lth layer in the network denoted by $\Psi_l(x)$ on input x, so that $\Psi_0(x) = x$, and Ψ_1 is the output of the first convolutional layer (or alternatively, the input to the 2nd layer). The loss is then averaged across the entire set $\mathbf{X}$ of N training samples.

We optimize with respect to each set of separable filters independently for each convolutional layer, resulting in four different optimization steps for each epoch for the entire network. We use the RMSProp [19] optimizer with a learning rate 0.001 and decay rate 0.95. We compare the accuracy of the separable network with the original filters throughout the optimization process and stop the optimization when the validation accuracy stops improving (about 1000 epochs in our case).

Table 1 depicts the development of SepConv accuracy as a function of intermediate feature maps K. It can be seen that the highest accuracy is already reached at $K = 7$. Considering the nearly linear increase in execution time, $K = 7$ becomes the value of choice in our later experiments and comparisons with other optimization methods.

1. github.com/Valentin4869/vcoptimizations
2. github.com/hegza/vcn-inference-rs

TABLE 1. INFERENCE CHARACTERISTICS OF SEPARABLE CONVOLUTION LAYER AS A FUNCTION OF FEATURE MAP COUNT K

K	Accuracy	Performance	Memory (KB)
1	87.3%	45 μs	1.9336
2	95.1%	48 μs	3.8672
3	96.6%	53 μs	5.8008
4	96.6%	58 μs	7.7344
5	97.0%	64 μs	9.668
6	97.1%	69 μs	11.6016
7	97.2%	75 μs	13.5352
8	97.1%	80 μs	15.4688
9	97.1%	85 μs	17.4023
10	97.1%	92 μs	19.3359
11	97.1%	97 μs	21.2695
12	97.3%	102 μs	23.2031
13	97.1%	109 μs	25.1367
14	97.3%	115 μs	27.0703
15	97.3%	121 μs	29.0039
16	97.4%	127 μs	30.9375

3.2. Pruning

Pruning is a method for reducing the number of weights in a network in the fully-connected layers [6], resulting in a sparse weights matrix, which allows the computation to be performed more efficiently using sparse matrix-vector multiplication [2]. We prune the network parameters in the lth layer that fall below the threshold $T = \frac{max(W^l)+min(W^l)}{2}$ by fixing them to 0. This initially results in a significant drop in accuracy compared to the baseline model. The network is then retrained for several epochs, which restores the accuracy of the network. A variant of this method [6] uses L2 regularization during the network training, which further sparsifies the layer weights.

We use the L2-regularized pruning approach where we first train the network normally, then we load the weights into a new model that is pruned and fine-tuned in two stages: first, the network is retrained for several (e.g. 30) epochs, where all the parameters are updated, using L2 regularization. The stage that follows removes all weights that are below T and then retrains (or fine-tunes) all the network parameters except for the ones removed in the pruning stage. This process is illustrated in Algorithm 1. We use a value of $\lambda = 0.35$ in our application for the L2-regularized loss function.

For inference, we implement a sparse matrix-vector multiplication procedure in a manner similar to [2]. The sparse weights matrix is reordered to the Compressed Sparse Row (CSR) format, which then allows the computation to be performed efficiently by eliminating all weights that have been pruned from the matrix.

3.3. Weights Quantization

Binarization is an optimization approach that reduces the precision of network weights and activations to 1-bit. The concept was first introduced and demonstrated in [8]. We follow this approach and replace all ReLU activations with the sign function defined as

$$sign(x) = \begin{cases} -1 & \text{if } x \leq 0 \\ +1 & \text{if } x > 0 \end{cases} \tag{3}$$

Additionally, we binarize all the weights in the network in the same way and use an approach identical to `vcbcnn` [12] to implement the binarized version of the vehicle classification network for inference.

For training the BCNN, following [7], we explicitly define the gradient of the *sign* function to be the identity function, such that $\frac{\partial sign(x)}{\partial x} = x$. Gradients are in full precision and are not binarized and the updates during training are done to the full-precision weights.

The binarized version is trained with the ADAM [13] optimizer. After training, only the binarized weights are used for inference. Binary-valued weights can be 'packed' such that each subgroup of 32 binary weights can occupy whole 32-bit registers, allowing for a reduction in storage size of the network parameters by up to 32×. This also allows for computing dot products much more efficiently as follows:

$$\mathbf{a} \cdot \mathbf{b} = \mathtt{W} - 2 \times \mathtt{popcount}(\mathtt{xor}(A, B)), \tag{4}$$

In Equation (4), both A and B are 32-bit unsigned integers containing the packed (decimal) representations of $\mathbf{a}$, $\mathbf{b} \in \{-1, +1\}^{\mathtt{W}}$, i.e., converted from an array of 32 bits to a 32-bit unsigned integer. The real-valued dot product is denoted by the $\cdot$ operator. The function `xor` is the bit-wise xor operation, and `popcount` is a function for computing the number of bits set to 1. The packing bitwidth `W` denotes the number of elements (or bits) that are packed together in a single unsigned integer register, which is 32 in our case.

Algorithm 1 Fully-connected layer pruning procedure

for l in Layers **do**
  for w in W^l **do**
    $w \leftarrow$ LoadPretrained(w)
for stage in Stages **do**
  // Retrain with L2 regularization
  for epoch in Train_Epochs **do**
    for l in Layers **do**
      for w in W^l **do**
        $w \leftarrow$ Update(w,η,$\frac{\partial \mathcal{L}(\mathbf{X})}{\partial w}$)
  // Pruning stage
  for l in DenseLayers **do**
    $T \leftarrow \frac{max(W^l)+min(W^l)}{2}$
    for w in W^l **do**
      if $w < T$ **then**
        $w \leftarrow 0$
  // Fine-tune parameters
  for epoch in Tuning_Epochs **do**
    for l in Layers **do**
      for w in NonZero(W^l) **do**
        $w \leftarrow$ Update(w,η,$\frac{\partial \mathcal{L}(\mathbf{X})}{\partial w}$)

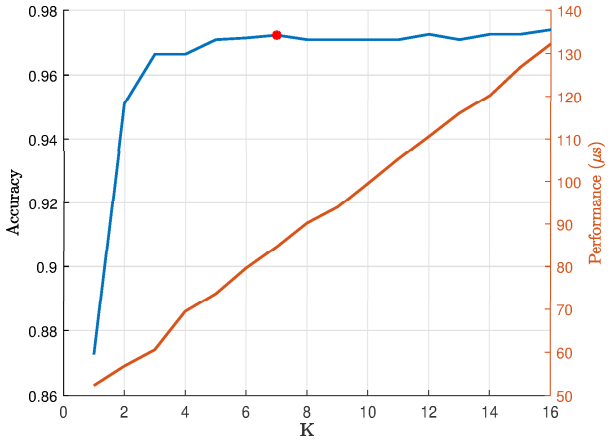

Figure 2. Performance/accuracy trade-off for different values of K.

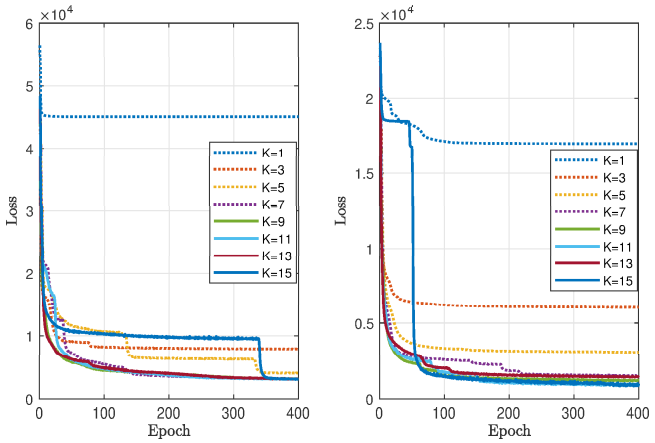

Figure 3. Loss for convolutional layers 1 (left) and 2 (right) shown for different values of K.

4. Results

This section presents the results of our experiments and compares their impacts on accuracy, performance, and memory usage for each optimization approach implemented. The two platforms used to perform the computation time measurements are provided in Table 2. From here on we refer to the platforms using their tag (GTX or Mali).

Table 1 shows the performance of convolution layers in the separable convolution network at different values of intermediate feature maps K. Performance is measured as execution time in microseconds on the GTX platform.

The graph in Figure 2 shows how the accuracy of the network changes as we change the number of intermediate featuremaps computed by the separable convolution layers. The performance of the convolutional layers for the whole network at each value of K is also shown in μs on the GTX platform. We find that $K = 7$ offers a good trade-off between accuracy and performance. While from Figure 2 it appears that for $K = 7$ the convergence is slower, the loss (Figure 3) is eventually reduced to a level comparable to other values above 7. The slow-down in convergence is not significant in our case, but it may impact larger networks and in cases where an optimizer other than RMSProp is used.

The plot of Figure 4 shows the effect of weight pruning to fully-connected layers. As the sparsity increases in the fully-connected layer, the network's accuracy begins to fluctuate at wider ranges; however, a high accuracy of 97% (marked in red) is reached at a sparsity level of nearly 99.9%. The fine-tuning stage, where pruned weights are held at 0 and are not updated, is shown in green. The blue plot marks the stage where all the weights are updated and the previously pruned weights in the previous step are restored and re-updated. All accuracy reports in Figure 4 are on the validation set.

A compact summary of these results is shown in Table 3. *Baseline (vnd. lib.)* refers to our implementation of the network using vendor-provided optimized libraries, which are cuDNN for GTX and Arm Compute Library for Mali.

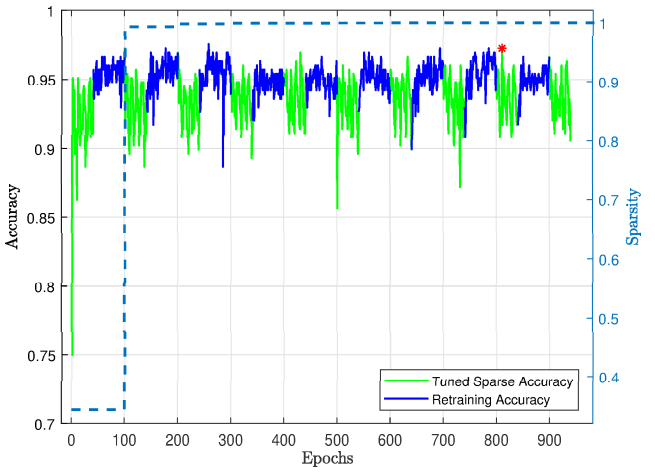

Figure 4. Accuracy of the network on the validation set at retraining stages and fine-tuning stages.

The column *Baseline (own)* is for our own low-level implementation without acceleration libraries.

We find that the best execution time is achieved through BCNN quantization on both platforms, but this approach also has the largest drop in accuracy across all methods. In contrast, on very low-resource devices (e.g. Internet of Things appliances), reduction of memory requirements can be more important than computation speed, and for this pruning outperforms BCNN by a clear margin. Combining pruning and separable convolution has the potential to further reduce the memory requirements, but the accuracy impact is not predictable and should be investigated.

5. Conclusion and Future Work

We compared several different CNN model optimization approaches for the application of vehicle classification. Our results show that significant reductions in memory requirements and computations can be achieved, however the compression technique must be selected based on the

TABLE 2. PLATFORMS USED FOR EXPERIMENTS

Tag	CPU	GPU	Operating System
GTX	Intel i7-7700K	NVidia GeForce GTX1080	Ubuntu 16.04
Mali	ARM Cortex-A72×2 + Cortex-A53×4	ARM Mali T860	Linux Firefly 4.4

TABLE 3. RESULTS OF OPTIMIZATION METHODS AT INFERENCE

	Baseline (vnd.lib.)	Baseline (own)	Pruning	SepConv	BCNN
GTX	0.4 ms	3.1 ms	0.7 ms	1.1 ms	**0.06** ms
Mali	30 ms	120 ms	66 ms	80 ms	**17.6** ms
Accuracy	97.5%	97.5%	95.5%	97.3%	94%
Memory (KB)	7350	7350	**150**	7254	230
Muls (in 10^6)	82.95	82.95	81.53	18.30	**0.43**
Adds (in 10^6)	86.12	86.12	84.35	20.37	**0.43**

optimization objective: BCNN quantization provides the highest performance, whereas pruning provides the highest reduction in storage requirements (assuming a network with significant fully-connected layers). In future works we plan to extend this study of optimization approaches to larger and more detailed datasets with real-life applications. Also, extending the analysis towards combinations of different compression methods in the spirit of [16] is definitely a direction worth investigating.

Acknowledgment

This work was funded by the Academy of Finland project 309903 CoEfNet.

References

[1] Alvarez J, Petersson L. DecomposeMe: Simplifying ConvNets for end-to-end learning. arXiv preprint arXiv:1606.05426, 2016 Jun 17.

[2] Bulu A, Fineman JT, Frigo M, Gilbert JR, Leiserson CE. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In Annual Symposium on Parallelism in Algorithms and Architectures (SPAA), 2009 (pp. 233-244).

[3] Ciregan D, Meier U, Schmidhuber J. Multi-column deep neural networks for image classification. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012 (pp. 3642-3649).

[4] Courbariaux M, Bengio Y, David JP. BinaryConnect: Training deep neural networks with binary weights during propagations. In Advances in Neural Information Processing Systems (NIPS), 2015 (pp. 3123-3131).

[5] Graves A, Jaitly N. Towards end-to-end speech recognition with recurrent neural networks. In International Conference on Machine Learning (ICML), 2014 (pp. 1764-1772).

[6] Han S, Pool J, Tran J, Dally W. Learning both weights and connections for efficient neural networks. In Neural Information Processing Systems (NIPS), 2015 (pp. 1135-1143).

[7] Hinton, G. Neural networks for machine learning. Coursera, video lectures, 2012.

[8] Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y. Binarized neural networks. In Advances in Neural Information Processing Systems (NIPS), 2016 (pp. 4107-4115).

[9] Huttunen H, Yancheshmeh FS, Chen K. Car type recognition with deep neural networks. In Intelligent Vehicles Symposium (IV), 2016 (pp. 1115-1120).

[10] Jaderberg M, Vedaldi A, Zisserman A. Speeding up convolutional neural networks with low rank expansions. In British Machine Vision Conference (BMVC), 2014.

[11] Kanda N, Takeda R, Obuchi Y. Elastic spectral distortion for low resource speech recognition with deep neural networks. In IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), 2013 (pp. 309-314).

[12] Khan M, Huttunen H, Boutellier J. Binarized convolutional neural networks for efficient inference on GPUs. In European Signal Processing Conference (EUSIPCO), 2018 (pp. 682-686).

[13] Kingma DP, Ba J. Adam: A method for stochastic optimization. In International Conference on Learning Representations (ICLR), 2015.

[14] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems (NIPS), 2012 (pp. 1097-1105).

[15] Lawrence S, Giles CL, Tsoi AC, Back AD. Face recognition: A convolutional neural-network approach. IEEE Transactions on Neural Networks, 1997, 8(1) (pp. 98-113).

[16] Lin JH, Xing T, Zhao R, Zhang Z, Srivastava M, Tu Z, Gupta RK. Binarized convolutional neural networks with separable filters for efficient hardware acceleration. In Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 (pp. 27-35).

[17] Parkhi OM, Vedaldi A, Zisserman A. Deep face recognition. In British Machine Vision Conference (BMVC), 2015.

[18] Rastegari M, Ordonez V, Redmon J, Farhadi A. Xnor-net: Imagenet classification using binary convolutional neural networks. In European Conference on Computer Vision (ECCV), 2016 (pp. 525-542).

[19] Tieleman T, Hinton G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. Coursera, video lectures, 2012.