FLATTENED CONVOLUTIONAL NEURAL NETWORKS FOR FEEDFORWARD ACCELERATION

Jonghoon Jin, Aysegul Dundar & Eugenio Culurciello
Purdue University, West Lafayette, IN 47907, USA
{jhjin, adundar, euge}@purdue.edu

ABSTRACT

We present flattened convolutional neural networks that are designed for fast feedforward execution. The redundancy of the parameters, especially weights of the convolutional filters in convolutional neural networks has been extensively studied and different heuristics have been proposed to construct a low rank basis of the filters after training. In this work, we train flattened networks that consist of consecutive sequence of one-dimensional filters across all directions in 3D space to obtain comparable performance as conventional convolutional networks. We tested flattened model on different datasets and found that the flattened layer can effectively substitute for the 3D filters without loss of accuracy. The flattened convolution pipelines provide around two times speed-up during feedforward pass compared to the baseline model due to the significant reduction of learning parameters. Furthermore, the proposed method does not require efforts in manual tuning or post processing once the model is trained.

1 INTRODUCTION

Recent success on fast implementation of convolutional neural networks (CNNs), and new techniques such as dropout enable researchers to train large networks that were not possible before. These large CNNs show great promise in visual and audio understanding which make them useful for applications in autonomous robots, security systems, mobile phones, automobiles and wearable supports. These applications require networks with high degree of accuracies, but also networks that can be executed in real-time. However, CNNs are computationally very expensive and require high performance servers or graphics processing units (GPUs).

To accelerate forward and backward passes of CNNs, there has been extensive work for efficient implementation of CNNs on GPUs (Krizhevsky et al., 2012; Chetlur et al., 2014) and CPUs (Vanhoucke et al., 2011), including linear quantization of network weights and inputs. For mobile platforms, like smartphones, computation of these big networks is still demanding and takes place on off-site servers because of their limited computing power and battery life. However, that requires a necessity to a reliable connectivity between the mobile device and off-site servers. Because this is not the case always, custom architectures have been explored for power and speed efficient implementation of CNNs (Jin et al., 2014; Merolla et al., 2014).

Another approach to speed up evaluation of CNNs is to reduce the number of parameters in the network representation. The work by Denil et al. (2013) is a good example to show that these networks have high redundancy in them. Considering that state of the art CNNs require hundreds of filters each layer and consist of three to five convolutional layers in general, finding essential representation with smaller parameters brings significant performance boost in terms of time and memory. Jaderberg et al. (2014); Denton et al. (2014) exploit the redundancy within convolutional layer after training and could obtain speedup by keeping the accuracy within 1% of the original models.

In this work, we take a similar approach to decrease the redundancy of the filters in convolutional neural networks. We achieve that in the training phase by separating the conventional 3D convolution filters into three consecutive 1D filters: convolution across channels (lateral), vertical and horizontal direction. We report similar or better accuracies on well-known datasets with the baseline network which has about ten times more parameters.

2 RELATED WORK

Convolutional Neural Networks (CNNs) exhibit high redundancy in the representation expressed as weight and bias parameters. The filters, visual interpretation of weights, in the network often have similar patterns and some of them have noise rather than distinct features. Having redundancy in the parameters not only degrades learning capacity of networks but accompanies unnecessary computations during feedforward pass as well as backpropagation. Many approaches have been proposed to find compact representation of CNNs by applying constraints on cost function or structure.

Sparsity in filters often help accelerate computations of filtering operation by simply skipping computations over non-zero values. Sparse feature learning aligned with findings in V1 neurons is proposed by Lee et al. (2007). By iterating L_1 and L_2 regularizer, this work successfully finds sparse and essential basis filters in over-complete system. However, size of the non-zero values in sparse filters is irregular and these non-zero values are located in arbitrary positions. Due to the arbitrary locations and shapes of sparse filters, in practice it is difficult to take advantage of sparsity with highly parallelized processing pipelines.

A classic but powerful method to accelerate filtering operations is to condition separability on object function (Rigamonti et al., 2013) so as to force the network to learn separable filters. In the literature, difficulties of optimization problem with L_1 penalty are relaxed. The separable 2D filter has a rank of one and makes the operation equivalent to two consecutive 1D convolutions, which significantly shortens the evaluation time of CNNs by an order of magnitude.

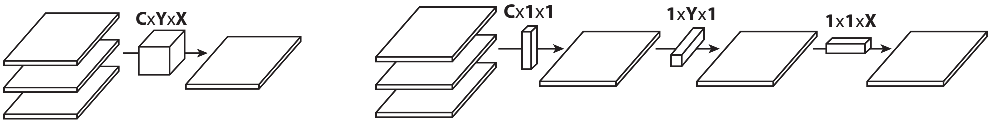
Recent works from Jaderberg et al. (2014); Denton et al. (2014) speed up CNNs evaluation time with low rank filter approximation. They compress convolutional layer of pre-trained networks by finding an appropriate low-rank approximation. Denton et al. (2014) extends the method to a large-scale task. Pre-trained 3D filters are approximated to low rank filters and the error is minimized by using clustering and post training to tune the accuracy. Their method demonstrates speedup of convolutional layers by a factor of two, while keeping the accuracy within 1% of the original model.

Another approach to reduce the parameters in CNNs is to explore the connections between layers. In the structure of state of the art CNNs, all convolutional planes are fully connected. Such connection scheme can handle and generalize all possible cases in training but in practice it is hard to learn sparse connectivity in output prediction. The importance of sparse connections in CNNs has been mentioned in the recent work (Szegedy et al., 2014). Also the connectivity previously was investigated by Culurciello et al. (2013), though many issues remain open for further research.

We apply structural constraints to conventional CNNs in order to learn 1D separated filters for feedforward acceleration. Our method does not alter training procedure of CNNs; backpropagating the error from output to the input along constrained paths. The approach bypasses difficulties in optimization problem witnessed in Rigamonti et al. (2013), but successfully learns 1D convolution filters. The proposed method does not require any manual tuning or changes in the structure once trained (Jaderberg et al., 2014; Denton et al., 2014), which simplify overall method. The concept of 1D convolution across channels is equivalent to the operation denoted as *mlpconv* layers in Network in Network (Lin et al., 2013). We also use 1×1 filters across channels to increase model discriminability for local patches within the receptive field. This layer also determines the number of filters in the subsequent layers because we add vertical and horizontal 1D filters which only performs convolution per channel. The difference between their and our work is that all of our filters are one-dimensional which provides significant reduction in parameters.

3 FLATTENING CONVOLUTION FILTERS

Similar to the notation used in Denton et al. (2014), weights in CNNs can be described as 4-dimensional filters: $W \in \mathbb{R}^{C \times X \times Y \times F}$, C is the number of input channels, X and Y are the spatial dimensions of the filter, and F is the number of filters or the number of output channels. Convolution



(a) 3D convolution

(b) 1D convolutions over different directions

Figure 1: The concept of 3D filter separation under rank-one assumption in the context of CNNs. Summation over all planes convolved with 2D filters produces a single output plane, which can be considered as 3D convolution. Three consecutive 1D filtering is an equivalent representation of 3D filter if its rank is one. C is the number of planes and its value is 3 in the diagram. Y and X denote filter height and width respectively. Bias is not considered here for simplicity.

for each channel output requires a filter $W \in \mathbb{R}^{C\times X\times Y}$ and is described as

$$F_f(x,y) = I * W_f = \sum_{c=1}^{C}\sum_{x'=1}^{X}\sum_{y'=1}^{Y} I(c, x-x', y-y')W_f(c,x',y') \tag{1}$$

assuming a stride of one where f is an index of output channel, $I \in \mathbb{R}^{C\times N\times M\times F}$ is the input map, N and M are the spatial dimensions of the input.

A rule of thumb to accelerate multi-dimensional convolution is to apply filter separation. Under rank-one assumption of the filter W_f, the unit rank filter $\hat{W}_f$ can be separated into cross-products of three one-dimensional filters as follows.

$$\hat{W}_f = \alpha_f \times \beta_f \times \gamma_f \tag{2}$$

We denote 1D convolution vectors as a lateral filter α_f : convolving features across channels; vertical filter β_f : across Y dimension; horizontal filter γ_f : across X dimension.

However, separability of filters is a strong condition and the intrinsic rank of filter W_f is higher than one in practice. As the difficulty of classification problem increases, the more number of leading components is required to solve the problem (Montavon et al., 2011). Learned filters in deep networks have distributed eigenvalues and applying the separation directly to the filters results in significant information loss.

Alternatively, we could restrict connections in receptive fields so that the model can learn 1D separated filters upon training. When applied to the equation 1, a single layer of convolutional neural networks is modified to

$$\hat{F}_f(x,y) = I * \hat{W}_f = \sum_{x'=1}^{X}\left(\sum_{y'=1}^{Y}\left(\sum_{c=1}^{C} I(c, x-x', y-y')\alpha_f(c)\right)\beta_f(x')\right)\gamma_f(y') \tag{3}$$

With this modification number of parameters to calculate each feature map decreases from XYC to $X+Y+C$, and number of the operations needed decreases from $MNCXY$ to $MN(C+X+Y)$.

Here we define flattened convolutional networks as CNNs whose one or more convolutional layer is converted to a sequence of 1D convolutions. We did not add the bias terms in the equations above to keep the equations clean. However; bias term is important for the training, and removing the bias terms in some of the 1D filters results in very slow learning. In our tests, we have separate bias terms for each three 1D filters.

4 Experimental Results

We tested the performance of the proposed model in alignment with a baseline model of CNNs on different classification tasks. In experiments, we used the Torch7 environment (Collobert et al., 2011) to demonstrate model performance as well as to handle customized gradient updates.

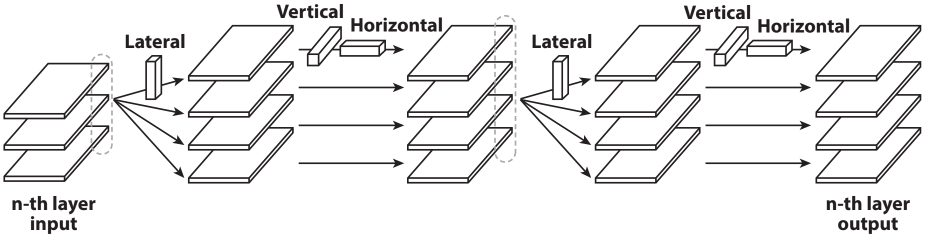


Figure 2: A single layer structure of flattened convolutional networks. Flattened layer includes l sets of 1D-separated convolutions over channels (lateral, L), vertical (V) and horizontal (H) direction. In this work, two stages of LVH combinations ($l = 2$) were chosen by cross-validation and it reported the same accuracy as measured in baseline model. V and H convolutions are operated in *full* mode to preserve the same output dimension as the baseline model. Bias is added after each of 1D convolution, but skipped in this illustration. No non-linear operator is applied within the flattened layer.

4.1 TRAINING BASELINE MODEL

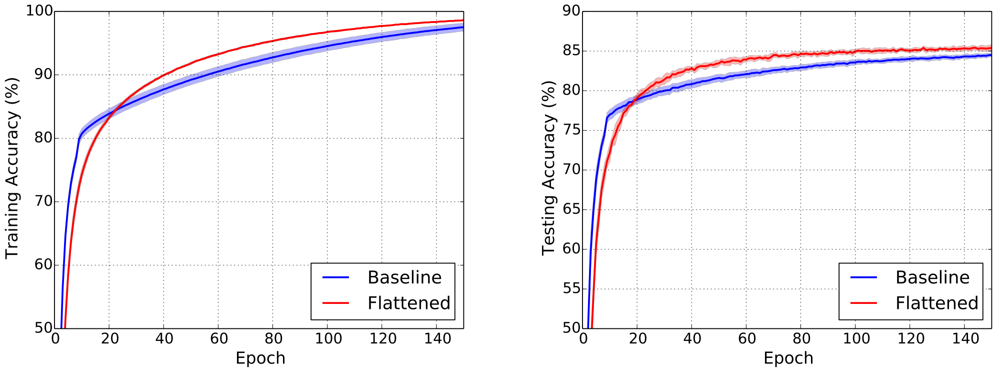
We choose a CNN model architecture same as the baseline model used in Srivastava & Salakhutdinov (2013) with a smaller multilayer perceptron. We keep the structure of CNNs to be generic so as to minimize unwanted interruption from hidden variables and make comparison to flattened model transparent. The model consists of 3 convolutional layers with 5×5 filters and double stage multilayer perceptron. The number convolutional filters in each layer are 96, 128 and 256 respectively, each layer includes a rectifier linear unit and a max-pooling with sizes of 2 and strides of 3. The two fully-connected multilayer perceptrons have 256 units each. The model is regularized with five dropout layers with the probability of 0.1, 0.25, 0.25, 0.5 and 0.5 respectively from lower to higher layer in order to prevent co-adaptation of features.

In our tests, we did not use data augmentation in order to concentrate learning capacity of models with respect to its structure. The training is initially set up with a learning rate of 0.1 and a momentum of 0.9. After the first eight epochs, the learning rate is reduced by one tenth. With the vanilla CNN and training configuration we were able to achieve performance comparable to state-of-the-art results on many datasets (see Table 2).

4.2 TRAINING FLATTENED MODEL

In flattened model, we use CNNs constructed with 1D filters as described in Figure 1. First, lateral filters (L) perform convolution across channels like *mlpconv* layers in Network in Network (Lin et al., 2013). Then, each channel is convolved with vertical and horizontal filters (V and H) whose filter sizes are $Y \times 1$ and $1 \times X$ in space respectively. While *mlpconv* applies 1D convolution across channels, this work extends 1D convolutions in space as well. Thus, the proposed method can be viewed as a generalization of training with 1D-separated filters in $\mathbb{R}^3$. Once the model structure is defined at the training stage, no post processing or fine-tuning is needed. The structural constraint forces the model to learn 1D separated filters, equivalently a rank-one 3D filter, except biases.

Replacing filters with dimension of $C \times X \times Y$ to filters dimensions of C, X, Y resulted in $2-3\%$ accuracy drops in our experiments on different datasets. 1D separated filters with dimensions of C, X and Y contains only 5% of the parameters as in 3D filter in commonly used CNNs (Krizhevsky et al., 2012; Sermanet et al., 2013). While Denil et al. (2013) demonstrates that 5% of essential parameters can predict the rest of parameters in the best case, such reduction in parameters accompanies accuracy loss (Gong et al., 2014; Lebedev et al., 2014). In our 1D training pipeline, parameters in one set of 1D filters are not enough to distinguish discriminate features, thus, results in failure to recover the target performance. We found that removing one direction from the LVH pipeline caused a significant accuracy loss, which implies that convolutions toward all directions are essential.



(a) Training accuracy

(b) Testing accuracy

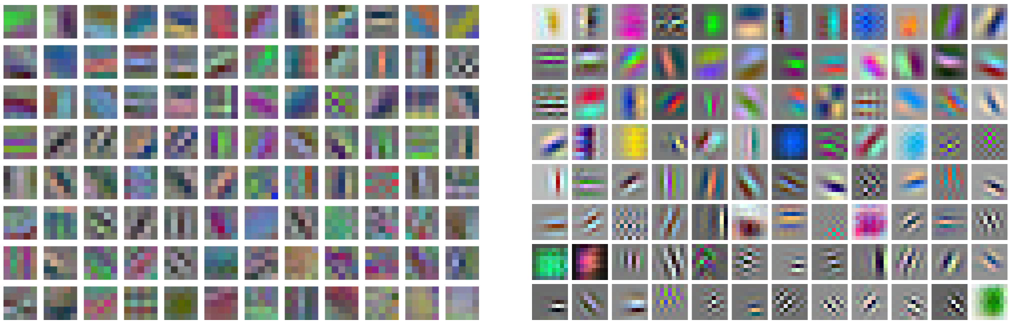
Figure 3: Convergence rate of flattened and baseline models both in training and testing. The structure of baseline and flattened model is specified in the section 4.1 and the table 1, respectively, and CIFAR-10 dataset is used in this experiment. The solid line denotes a mean and the shade around the line indicates a standard deviation of the curve. The variation of training for the flattened model is too small to appear in the illustration.

We cascaded a set of Lateral-Vertical-Horizontal (LVH) convolutional layers to compensate accuracy loss at the cost of using more parameters. Empirical results show that two cascaded sets of LVH layers achieve the same performance as a standard convolution layer in the baseline model. The flattened layer, consisting of two stages of LVH, is illustrated in Figure 2 and used throughout the experiments. The V and H convolutions within the flattened layer are operated in *full* mode in order not to lose boundary features and to keep output dimensions the same as the baseline model. With this method, we were able to decrease the number of parameters $8 - 10\times$ compared to that in the baseline model. Different number of LVH layers could be cascaded depending on the difficulties of classification tasks and the effect of parameter reduction is discussed in the section 4.3. We used the same training configuration of baseline model except decreased weight decay to give more freedom to adapt features.

The serialized model with 1D convolutions is more vulnerable to vanishing gradient problem than standard CNNs. Attaching many 1D convolution layer provides more flexibility in filter shape therefore helps draw delicate decision boundary. However, longer gradient path experiences more steps of parameter updates and error accumulation, which possibly cause fast decaying gradients. This trend is more persistent for V or H convolution because they get feedback from only one channel. From the gradients update $\delta x_i = \sum_{k=1}^{K} w_k \delta x_{i+1}$, accumulation of gradients and attenuation by weights are balanced each other in the standard CNNs. However, the flattened structure has generally few connections in H and V convolution. Vanishing gradients can be handled with smart weight initialization. Normalized initialization balanced with forward and backward passes (Glorot & Bengio, 2010) or constant error propagation helps deliver gradients to lower layers. In our experiments, they yielded more successful training than the heuristic (LeCun et al., 1998b) while the baseline model with the highest accuracy was trained with the heuristic initialization.

With the proper weight initialization, the flattened model achieves the comparable accuracy as the baseline model. Figure 3 reports the training and testing accuracy on the CIFAR-10 dataset. Considering that learning rate for the baseline model is decreased in order to achieve the highest accuracy, the baseline model saturates earlier and to a lower accuracy compared to the flattened model. The shaded region in the plots indicates a variation of accuracies obtained from samples with different random seeds. The learning curve of the flattened model shows consistent results with less variation. The flattened method alleviates the training effort by accelerating backpropagation and is presented in the section 4.6.

The first layer filters trained on CIFAR-10 within the flattened structure are reconstructed to 3D filters and presented in Figure 4 though the first layer is not converted in the main experiment (see Section 4.3). Filters are reconstructed by cross-product of 1D filters trained from two sets of LVH



(a) Filters in baseline model

(b) Reconstructed filters in flattened model

Figure 4: Visualization of the first layer filters trained on CIFAR-10. Filters are reconstructed from cross-product of 1D convolution filters and contains clear and diverse features. Filters are sorted by variance in descending order and bias is excluded during reconstruction.

convolution layers. The richness of distinct features in the filters is necessary for model discriminability. Surprisingly, the reconstructed filters have distinct Gabor edges and color blobs. Features have high contrast and edges are sparse as in Lee et al. (2007) without L_1 penalty. This finding supports the effectiveness of the method and explains the comparable performance of proposed model over the baseline model.

4.3 Parameter Reduction

Flattened model applied to CNNs generally relaxes computational demands by reducing the number of parameters. Here we analyze parameter reduction and its trade-off in practical viewpoint.

Flattened convolutional layer used in this work has two stages of LVH convolutions. Corresponding filter dimensions is $F(F + C + 2X + 2Y)$ while the regular 3D filters in the baseline model has dimension of $CXYF$, where C and F denote the number of input and output channels, X and Y are the spatial dimensions of the filter.

Considering the filter dimensions are relatively small and fixed throughout layers, our gain for parameter reduction is mostly depend on the ratio between F and C. We denote the ratio $k = F/C$ and we want to flatten a convolutional layer only if k satisfies the equation 4

$$C^2k^2 + C^2k + 2C(X + Y) < C^2XYk \tag{4}$$

where the left side denotes the computions required for a flattened layer and the right side for a standard convolution layer.

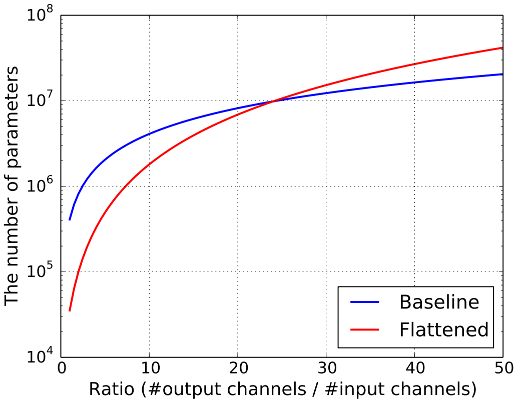


Figure 5: Filter dimensions of $X = Y = 5$ and $C = 128$ are used to observe reduction efficiency.

Figure 5 visualizes the relationship between the parameter reduction of flattened network compare to baseline network. The flattening method is guaranteed to reduce a large portion of parameters as long as the number of channels increases smoothly. Most layers of CNNs can benefit from this method since the number of channels does not decrease over layers and the ratio between channels usually resides between 1 and 3, other than the first layer (Krizhevsky et al., 2012; Sermanet et al., 2013; Szegedy et al., 2014; Simonyan & Zisserman, 2014). First layer begins with three channels, so performing 96 filters gives us a ratio of 32 where the baseline model has less parameters. For the purpose of parameter reduction, we selectively applied flattening to the second and third layers of the baseline CNN though we successfully applied the flattening to the all convolutional layers and achieved the same accuracy. Table 1 summarizes details of two CNN models in terms of the number of parameters.

Table 1: The number of parameters in each layer of flattened and baseline CNN model.

	Baseline Model	**Flattened Model**	**Reduction**
Layer 1 Parameters	$7,200$	Not applied [1]	0.0%
Layer 2 Parameters	$307,200$	$30,912$	89.9%
Layer 3 Parameters	$819,200$	$102,144$	87.5%

4.4 Memory Usage

In this section we compare the memory consumption of the baseline and flattened network in training. It is an important concern since memory limit could affect the degree of parallelism as the model scales up. The flattened layer needs to hold all intermediate states for the backward pass whose size is as big as the size of output planes. Considering that each convolution is broken down into N pieces of 1D convolution, the flattened structure produces $N - 1$ intermediate states that needed to be stored in the memory. This is true for the naïve implementation of convolution as seen in figure 6 where it uses nested for-loops and is optimized in memory usage.

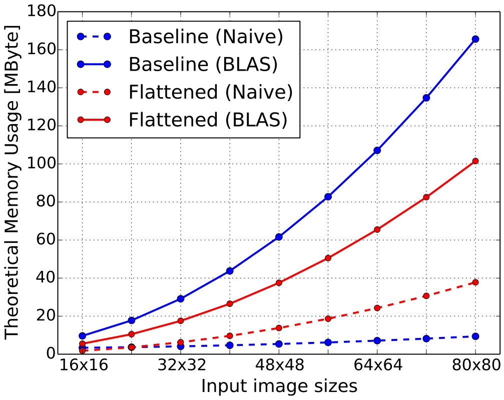


Figure 6: Theoretical memory usage for a single flattened layer. Memory usage optimized implementation (Naïve) and performance optimized implementation (BLAS) are presented.

However, BLAS-friendly convolution routine is generally used to achieve the highest performance in time and adapted in scientific computing framework (Collobert et al., 2011; Jia et al., 2014; Chetlur et al., 2014). The naïve approach exploits limited parallelism due to its frequent memory access, which throttles the real-time performance of CNNs on many platforms. The BLAS implementation for the baseline model handles convolution as a matrix multiplication at the cost of additional memory copy and space. On the other hand, 1D convolution pipeline in flattened layer can achieve sufficient parallelism without additional resources in feedforward and with little extra memory in backpropagation as opposed to the 3D convolution. Therefore, the flattened layer uses less memory than a baseline convolution layer in practice even though each convolution is broken down into 6 pieces as illustrated in figure 6.

4.5 Classification Accuracy

Despite of the reduced number of parameters, the flattened model does not suffer from accuracy loss. We tested the model on CIFAR-10, CIFAR-100 and MNIST datasets.

The CIFAR-10 and CIFAR-100 (Krizhevsky & Hinton, 2009) are datasets of 32×32 RGB images and often used as a standard measurement to evaluate classification capability. CIFAR-10 has 10 classes while CIFAR-100 has 100 classes. Both datasets consist of with $50,000$ training and $10,000$ testing images. Before use, datasets are preprocessed with contrast normalization per image followed by ZCA whitening to the whole dataset as Goodfellow et al. (2013). Both models tend to reach the same performance but in our experiments the accuracy of flattened model outperforms the baseline model slightly as can be seen from Table 2.

[1] The flattening is not applied to the first layer for purpose of parameter reduction based on figure 5 though the technique in the first layer achieved comparable accuracy as well.

The MNIST dataset (LeCun et al., 1998a) consists of hand written digits of 0-9. This dataset contains $60,000$ training and $10,000$ testing images. We applied contrast normalization per image as Goodfellow et al. (2013) without ZCA whitening since hand-written images have low cross-correlation values as opposed to CIFAR datasets. The classification on MNIST is relatively easier than CIFAR datasets and the accuracies of two models are highly saturated. Whereas it is difficult to compare models because of the simplicity of the dataset, baseline and flattened models give almost the same accuracies.

Table 2: Classification accuracy of baseline and flattened model on different datasets

Dataset	**Model Type**	**Test Accuracy**
CIFAR-10	Baseline Model	86.42%
	Flattened Model	87.04%
CIFAR-100	Baseline Model	60.08%
	Flattened Model	60.92%
MNIST	Baseline Model	99.62%
	Flattened Model	99.56%

4.6 Acceleration

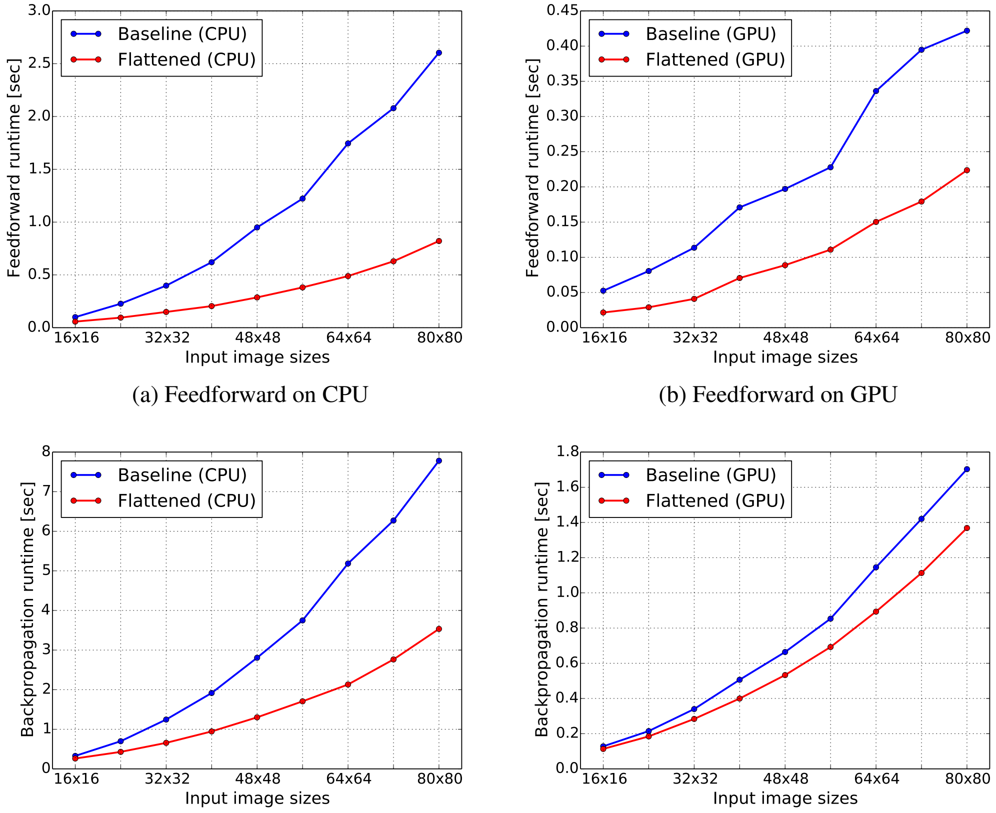
Due to the reduced parameters in 1D convolution pipelines, flattened structure reduces computational demands of CNNs thus accelerate both feedforward and backward computations of CNNs. Profiling results of flattened and baseline models for feedforward and backpropagation passes are presented in Figure 7. We ran the same tests on CPU and GPU to check performance of flattened structure versus different degree of parallelism. The performance was measured on Intel i7 3.3GHz CPU and NVIDIA Tesla K-40 GPU. Different image sizes from 16×16 to 80×80 with 128 batch were applied to the models. We used a single convolution layer for baseline model which has dimensions of $128 \times 128 \times 5 \times 5$ filters, and the corresponding 6 pieces of convolution layer for flattened model which has 2 sets of filters with dimensions of $128 \times 128 \times 1 \times 1$, $128 \times 128 \times 5 \times 1$ and $128 \times 128 \times 1 \times 5$ filters.

Feedforward pass is the part that mostly benefits from the manipulation of 1D convolution filters. The flattened layer runs about two times faster than the conventional convolutional layer during forward pass on both platforms. The acceleration tends to increase as the size of images gets larger because overhead time becomes negligible for large size of images. We implemented L, V and H convolution routines that are optimized in speed. It is evident that the filter separation reduces the absolute number of computations, which provides the acceleration. Efficient memory access is another factor for feedforward acceleration. In the 1D convolution pipeline, convolution is processed along one direction only, therefore input pixels can be considered as a 1D image sequence, which minimizes effort in data index and access. The results imply that the flattened structure is more promising if used in the lower layers of CNNs where image size is not small and the ratio between input/output channels increases smoothly.

Using flattened layer also reduces training time on CPU and GPU. The backpropagation consists of gradient propagation and parameter update. In the former case, its computation benefits from the coalesced memory access of 1D convolution pipeline as in feedforward. However, the latter requires data accumulation over all pixels at each 1D convolutional layer. Therefore the serial operation with frequent global memory access make the acceleration negligible on GPU.

5 Conclusion

In this work, we propose a flattening technique to reduce the number of parameters in convolutional neural networks for feedforward acceleration. We convert each of convolutional layer of 3D



(a) Feedforward on CPU

(b) Feedforward on GPU

(c) Backpropagation on CPU

(d) Backpropagation on GPU

Figure 7: Speed-up comparison for feedforward and backpropagation execution time measured on CPU and GPU. Input images with 128 batch size were applied to a single flattened layer, which is equivalent to 128-to-128 convolution with 5×5 filter in terms of classification accuracy. Flattened model runs faster on both platforms and the performance gap increases as the size of image is larger. Profiling results were measured on BLAS-accelerated convolution routines for all experiments. More optimized CUDA implementation for 1D convolution would shorten execution time on GPU.

convolution pipeline into a sequence of 1D convolutions across channels, vertical and horizontal directions in training phase. We found that the model molded in 1D structure is able to learn 1D filters successfully and achieves about two times speed-up in evaluation compared to the baseline model. Furthermore, with ten times less parameters, the flattened convolutional networks achieve similar or better accuracies on CIFAR-10, CIFAR-100, and MNIST. In addition, the proposed method does not require efforts in manual tuning or post processing once the model is trained and it bypasses the difficulties in solving optimization problem to learn 1D filters. The simple nature of the parameter reduction method could be applied to accelerate a very large-scale model as well and this remains as future work.

ACKNOWLEDGMENTS

This work is supported by Office of Naval Research (ONR) grants 14PR02106-01 P00004 and MURI N000141010278. We gratefully appreciate the support of NVIDIA Corporation with the donation of GPUs used for this research.

REFERENCES

Chetlur, Sharan, Woolley, Cliff, Vandermersch, Philippe, Cohen, Jonathan, Tran, John, Catanzaro, Bryan, and Shelhamer, Evan. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

Collobert, Ronan, Kavukcuoglu, Koray, and Farabet, Clément. Torch7: A matlab-like environment for machine learning. In *BigLearn, Neural Information Processing Systems Workshop*, 2011.

Culurciello, Eugenio, Jin, Jonghoon, Dundar, Aysegul, and Bates, Jordan. An analysis of the connections between layers of deep neural networks. *arXiv preprint arXiv:1306.0152*, 2013.

Denil, Misha, Shakibi, Babak, Dinh, Laurent, de Freitas, Nando, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pp. 2148–2156. 2013.

Denton, Emily L, Zaremba, Wojciech, Bruna, Joan, LeCun, Yann, and Fergus, Rob. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pp. 1269–1277. 2014.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In Teh, Yee W. and Titterington, D. M. (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-10)*, volume 9, pp. 249–256, 2010.

Gong, Yunchao, Liu, Liu, Yang, Ming, and Bourdev, Lubomir D. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Jaderberg, Max, Vedaldi, Andrea, and Zisserman, Andrew. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Jin, Jonghoon, Gokhale, Vinayak, Dundar, Aysegul, Krishnamurthy, Bharadwaj, Martini, Berin, and Culurciello, Eugenio. An efficient implementation of deep convolutional neural networks on a mobile coprocessor. In *Circuits and Systems (MWSCAS), 2014 IEEE 57th International Midwest Symposium on*, pp. 133–136. IEEE, 2014.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Lebedev, Vadim, Ganin, Yaroslav, Rakhuba, Maksim, Oseledets, Ivan V., and Lempitsky, Victor S. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998a.

LeCun, Yann, Bottou, Léon, Orr, Genevieve B., and Müller, Klaus-Robert. Effiicient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pp. 9–50, London, UK, UK, 1998b. Springer-Verlag. ISBN 3-540-65311-2.

Lee, Honglak, Battle, Alexis, Raina, Rajat, and Ng, Andrew Y. Efficient sparse coding algorithms. In Schölkopf, B., Platt, J.C., and Hoffman, T. (eds.), *Advances in Neural Information Processing Systems*, pp. 801–808. MIT Press, 2007.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Merolla, P.A., Arthur, J.V., Alvarez-Icaza, R., Cassidy, A.S., Sawada, J., Akopyan, F., Jackson, B.L., Imam, N., Guo, C., Nakamura, Y., Brezzo, B., Vo, I., Esser, S.K., Appuswamy, R., Taba, B., Amir, A., Flickner, M.D., Risk, W.P., Manohar, R., and Modha, D.S. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, pp. 668–673, August 2014.

Montavon, Grégoire, Braun, Mikio, and Müller, Klaus-Robert. Kernel analysis of deep networks. *Journal of Machine Learning Research*, 12:2563–2581, 2011.

Rigamonti, R., Sironi, A., Lepetit, V., and Fua, P. Learning separable filters. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pp. 2754–2761, June 2013.

Sermanet, Pierre, Eigen, David, Zhang, Xiang, Mathieu, Michaël, Fergus, Rob, and LeCun, Yann. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Srivastava, Nitish and Salakhutdinov, Ruslan. Discriminative transfer learning with tree-based priors. In *Advances in Neural Information Processing Systems*, pp. 2094–2102. 2013.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

Vanhoucke, Vincent, Senior, Andrew, and Mao, Mark Z. Improving the speed of neural networks on CPUs. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.