SELECTING OPTIMAL LAYER REDUCTION FACTORS FOR MODEL REDUCTION OF DEEP NEURAL NETWORKS

Lukas Mauch and Bin Yang

Institute of Signal Processing and System Theory, University of Stuttgart, Germany

ABSTRACT

Deep neural networks (DNN) achieve very good performance in many machine learning tasks, but are computationally very demanding. Hence, there is a growing interest on model reduction methods for DNN. Model reduction allows to reduce the number of computations needed to evaluate a trained DNN without a significant performance degradation. In this paper, we study layerwise reduction methods that reduce the number of computations in each layer independently. We consider the pruning and low-rank approximation method for model reduction. Up to now, often a constant reduction factor is used in all layers. In this paper, we show that a non-uniform allocation of reduction factors to different layers can greatly improve the performance of the reduced DNN. For this purpose, we select the optimal layer reduction factors in terms of an optimization problem. Experiments on three different benchmark datasets demonstrate the superior performance of our method.

Index Terms— Deep neural networks, model reduction, pruning, low-rank approximation

1. INTRODUCTION

Today deep neural networks (DNN), though very powerful, are limited to applications with enough computation power and memory because evaluating a trained DNN is computationally demanding [1, 2]. As DNN is the state-of-the-art method for many machine learning tasks [1, 2, 3, 4], there is a growing interest in how to reduce its complexity without degrading the performance. This is an important step to make DNN applicable even on mobile or embedded devices.

Model reduction means to transform a trained network with the parameter vector $\underline{\theta}$ to a reduced network with the new parameter vector $\underline{\tilde{\theta}} = R(\underline{\theta}, \alpha_{tot}, \{\underline{x}_0(n)\}, \{\underline{y}(n)\})$ in such a way that the reduced network requires less computations in evaluation. The transform depends on the desired total reduction factor α_{tot}, i.e. the percentage of the saved computations for the whole network. It may also depend on a sequence of input data $\{\underline{x}_0(n)\}$ and optionally a sequence of reference output data $\{\underline{y}(n)\}$.

Most proposed methods work layerwise without reference, i.e. the parameter vector $\underline{\theta}_l$ of layer l is transformed to $\underline{\tilde{\theta}}_l = R_l(\underline{\theta}_l, \alpha_l, \{\underline{x}_0(n)\})$ independent of the other layers, where α_l is the reduction factor of layer l [5, 6, 7].

In this paper, we consider two model reduction techniques, pruning and low-rank approximation. The former deletes some selected neurons in each layer according to some selection strategies and reduces the number of parameters [5, 6]. The latter replaces the weight matrix in each layer by a low-rank approximation [7]. This is beneficial because the product of the weight matrix and input vector can be split into two matrix-vector-products with smaller matrices.

Using layerwise reduction methods, a central problem is how to choose the reduction factors α_l of individual layers, called layer reduction factors. Most works on model reduction of DNN use an iterative approach switching between network reduction and supervised retraining. In each iteration, the reduction factor for each layer is chosen heuristically based on the activation of the neurons for a given labeled training sequence or comparing the magnitude of the weights against a threshold [5, 8, 9]. The iterations are stopped if the desired total reduction factor α_{tot} for the whole network is reached or the network can not be reduced further. These methods need labeled reference data and, due to the iterative reduction and retraining, are computationally demanding.

We look at this problem from a different perspective and study how to choose the layer reduction factors α_l to satisfy the requirement of a total reduction factor α_{tot} without iterative retraining. This problem has not been addressed in the literature. A careful choice of α_l is obviously necessary, since sparsity and low-rank structure of the weight matrices may change from layer to layer. This depends on the individual layer width and also on the application at hand.

In this paper, we do not study how to do the layerwise model reduction. We rather focus on how to choose the layer reduction factors α_l in an optimal way to get a reduced network with a given total reduction factor α_{tot} and without a significant performance degradation. In experiments, we test our method on networks trained on benchmark datasets MNIST, SVHN and NORB and demonstrate the significant performance improvement by an optimal allocation of layer reduction factors over a constant reduction factor.

2. LAYERWISE NETWORK REDUCTION

2.1. Problem formulation

We consider the model reduction of a trained fully connected DNN. The full DNN consists of one input layer of neurons $\underline{x}_0 \in \mathbb{R}^{M_0}$, $L-1$ layers of hidden neurons $\underline{x}_l \in \mathbb{R}^{M_l}$ $(1 \le l < L)$ and one output layer of neurons $\underline{x}_L \in \mathbb{R}^{M_L}$. Equivalently, the DNN has L layers of weights or nonlinear transforms

$$\underline{x}_l = f_l(\underline{x}_{l-1}, \underline{\theta}_l) = \Phi_l\left(\underline{a}_l\right), \quad (1)$$

$$\underline{a}_l = \mathbf{W}_l \underline{x}_{l-1} + \underline{b}_l, \ 1 \le l \le L. \quad (2)$$

$\underline{a}_l \in \mathbb{R}^{M_l}$ is the activation, $\mathbf{W}_l \in \mathbb{R}^{M_l \times M_{l-1}}$ the weight matrix, $\underline{b}_l \in \mathbb{R}^{M_l}$ the bias vector and $\Phi_l(\cdot)$ the nonlinear activation function of layer l. The parameter vector of layer l is $\underline{\theta}_l \in \mathbb{R}^{M_l(M_{l-1}+1)}$ and contains all elements of $\mathbf{W}_l$ and $\underline{b}_l$. The number of multiplications at layer l is $M_{l-1}M_l$ [1].

After the model reduction, the reduced network computes

$$\tilde{\underline{x}}_l = \tilde{f}_l(\tilde{\underline{x}}_{l-1}, \tilde{\underline{\theta}}_l) = \Phi_l\left(\tilde{\underline{a}}_l\right) \quad (3)$$

$$\tilde{\underline{a}}_l = \tilde{\mathbf{W}}_l \tilde{\underline{x}}_{l-1} + \tilde{\underline{b}}_l, \ 1 \le l \le L, \quad (4)$$

where $\tilde{\underline{\theta}}_l$ is the new parameter vector of the reduced layer l. Layerwise network reduction means that $\tilde{\underline{\theta}}_l$ is determined by $\underline{\theta}_l$ only and not by the other layers. In this paper, when we speak of layer l, we mostly mean the layer l of weights as described in Eq. 1 - 4 and not a layer of neurons.

2.2. Naive pruning

The basic idea of pruning is to delete some irrelevant neurons in each layer. Assume that we keep only $\tilde{M}_l$ from M_l neurons in $\underline{x}_l$ with $\tilde{M}_l < M_l, \ 0 \le l < L$. The number of output neurons remains unchanged $\tilde{M}_L = M_L$. In this case, $\tilde{\underline{x}}_{l-1} \in \mathbb{R}^{\tilde{M}_{l-1}}$ and $\tilde{\mathbf{W}}_l \in \mathbb{R}^{\tilde{M}_l \times \tilde{M}_{l-1}}$ and Eq. 4 requires $\tilde{M}_{l-1}\tilde{M}_l$ multiplications instead of $M_{l-1}M_l$. This corresponds to a reduction factor of

$$\alpha_l = 1 - \frac{\tilde{M}_{l-1}\tilde{M}_l}{M_{l-1}M_l} \quad (5)$$

for layer $1 \le l < L$ and

$$\alpha_L = 1 - \frac{\tilde{M}_{L-1}M_L}{M_{L-1}M_L} = 1 - \frac{\tilde{M}_{L-1}}{M_{L-1}} \quad (6)$$

for the last layer L. The total reduction factor of the whole network is

$$\alpha_{tot} = 1 - \frac{\sum_{l=1}^{L-1} \tilde{M}_{l-1}\tilde{M}_l + \tilde{M}_{L-1}M_L}{\sum_{l=1}^{L} M_{l-1}M_l}. \quad (7)$$

If we use a constant layer reduction factor $\alpha_1 = ... = \alpha_L = \alpha$ in all layers, called uniform allocation of reduction factors, the same total reduction factor $\alpha_{tot} = \alpha$ yields. Note that in the pruning case, a reduction of the number of input and output neurons for a layer has a quadratic influence to α_l because the weight matrix $\mathbf{W}_l$ is shrinked in both height and width.

There are different approaches to select the neurons to be pruned and to calculate $\tilde{\mathbf{W}}_l, \tilde{\underline{b}}_l$ from $\mathbf{W}_l, \underline{b}_l$. Since this is not the main focus of this paper, we use a simple naive pruning method to demonstrate the impact of optimum allocation of reduction factors α_l. By using a sequence of N unlabeled input data $\{\underline{x}_0(n)\}$, we calculate the sample covariance matrices of the input $\underline{x}_{l-1}(n)$ of all L layers:

$$\mathbf{C}_{l-1} = \frac{1}{N}\sum_{n=1}^{N}(\underline{x}_{l-1}(n) - \underline{\mu}_{l-1})(\underline{x}_{l-1}(n) - \underline{\mu}_{l-1})^T, \quad (8)$$

$$\underline{\mu}_{l-1} = \frac{1}{N}\sum_{n=1}^{N}\underline{x}_{l-1}(n), \ 1 \le l \le L. \quad (9)$$

Then, we keep these $\tilde{M}_{l-1}$ neurons in $\underline{x}_{l-1}$ with the largest diagonal elements in $\mathbf{C}_{l-1}$. The remaining $M_{l-1} - \tilde{M}_{l-1}$ neurons are deleted.

2.3. Low-rank approximation

The basic idea of low-rank approximation of $\mathbf{W}_l \in \mathbb{R}^{M_l \times M_{l-1}}$ in Eq. 2 is the calculation of $\tilde{\mathbf{W}}_l = \mathbf{A}_l\mathbf{B}_l$ from $\mathbf{W}_l$ by a principal component analysis (PCA) with $\mathbf{A}_l \in \mathrm{R}^{M_l \times k_l}$ and $\mathbf{B}_l \in \mathrm{R}^{k_l \times M_{l-1}}$, $k_l < \min(M_{l-1}, M_l)$. The matrix-vector-product $\mathbf{W}_l\underline{x}_{l-1}$ involving $M_{l-1}M_l$ multiplications is then split into two products with $M_{l-1}k_l$ and M_lk_l multiplications, respectively. The reduction factor in layer l is thus

$$\alpha_l = 1 - \frac{(M_{l-1} + M_l)k_l}{M_{l-1}M_l}, \ 1 \le l \le L. \quad (10)$$

The total reduction factor of the whole network is

$$\alpha_{tot} = 1 - \frac{\sum_{l=1}^{L}(M_{l-1} + M_l)k_l}{\sum_{l=1}^{L} M_{l-1}M_l}. \quad (11)$$

Again, $\alpha_{tot} = \alpha$ follows from $\alpha_1 = ... = \alpha_L = \alpha$. Note that in contrast to pruning, a reduction of the number of principal components has only a linear impact to α_l because $\mathbf{A}_l$ and $\mathbf{B}_l$ are shrinked only in one dimension.

3. OPTIMAL ALLOCATION OF LAYER REDUCTION FACTORS

3.1. Optimization problem

The easiest way to choose α_l is the uniform allocation $\alpha_1 = ... = \alpha_L = \alpha$. According to the previous section, the total reduction factor is $\alpha_{tot} = \alpha$ as well.

An optimal allocation of the layer reduction factors means to determine $\alpha_1, ..., \alpha_L$ for a given value of α_{tot} such, that the performance of the reduced DNN is optimum in some sense.

[1] In this paper, we use the number of multiplications to quantify the computational complexity and reduction factor. This means, we neglect the addition of the bias term $\underline{b}_l$ and the calculation of the activation function Φ_l.

Let $\underline{\vartheta} = [\tilde{M}_0, ..., \tilde{M}_{L-1}]^T \in \mathbb{N}^L$ and $\underline{\vartheta} = [k_1, ..., k_L]^T \in \mathbb{N}^L$ be the numbers of selected neurons in $\underline{x}_0, ..., \underline{x}_{L-1}$ for the pruning case and the number of selected principal components in $\mathbf{W}_1, ..., \mathbf{W}_L$ for the low-rank approximation, respectively. According to Eq. 5,6 and 10, there is an one-to-one unique relationship between $\{\alpha_l\}$ and $\underline{\vartheta}$. Hence, we can equivalently optimize $\underline{\vartheta}$ for a given minimum total reduction factor $0 < \alpha_{tot,min} < 1$.

Our basic idea is to choose $\underline{\vartheta}$ such that the approximation errors induced by the model reduction are minimized in some sense. One possibility is to minimize the squared error $\|\underline{\tilde{x}}_L - \underline{x}_L\|^2 = \|\tilde{f}(\underline{x}_0, \underline{\tilde{\theta}}, \underline{\vartheta}) - f(\underline{x}_0, \underline{\theta})\|^2$ between the output of the reduced and full DNN. This is, however, not practical because it requires the calculation of the reduced DNN $\tilde{f}(\underline{x}_0, \underline{\tilde{\theta}}, \underline{\vartheta})$ for all possible values of the integer vector $\underline{\vartheta}$.

We propose a layerwise cost function

$$J(\underline{\vartheta}) = \sum_{l=1}^{L} J_l(\vartheta_l) \tag{12}$$

to be minimized. $J_l(\vartheta_l)$ defines a normalized approximation error induced by model reduction at layer l with $\vartheta_l = \tilde{M}_{l-1}$ for pruning and $\vartheta_l = k_l$ for low rank approximation. The sum of $J_l(\vartheta_l)$ over all layers is important because we applied layerwise reduction methods and each $J_l(\vartheta_l)$ represents only the reduction error of one layer and not that of the whole network.

3.2. Low-rank approximation

In case of low-rank approximation, $J_l(\vartheta_l) = J_l(k_l)$ can be easily calculated from the singular value decomposition (SVD) of the weight matrix $\mathbf{W}_l$ of the full network without any additional input data. Let $\sigma_1 \geq ... \geq \sigma_{k_L} \geq 0$ with $K_l = \min(M_{L-1}, M_L)$ be the singular values of $\mathbf{W}_l$ in a decreasing order. In low-rank approximation $\mathbf{W}_l$ is approximated by a rank k_l matrix $\tilde{\mathbf{W}}_l$ with the same singular vectors as $\mathbf{W}_l$ and the singular values $\sigma_1, ..., \sigma_{K_l}, 0, ..., 0$. We propose

$$J_l(\vartheta_l) = J_l(k_l) = \frac{\|\mathbf{W}_l - \tilde{\mathbf{W}}_l\|_F^2}{\|\tilde{\mathbf{W}}_l\|_F^2} = \frac{\sum_{i=k_l+1}^{K_l} \sigma_i^2}{\sum_{i=1}^{k_l} \sigma_i^2}, \tag{13}$$

where $\|\cdot\|_F$ denotes the Frobenius norm.

3.3. Pruning

For the pruning method, we need a sequence of N unlabeled input samples $\{\underline{x}_0(n)\}$ to calculate $\{\underline{x}_{l-1}(n)\}$ of all layers of the full DNN and their corresponding sample covariance matrices $\mathbf{C}_{l-1}$, $1 \leq l \leq L$ as in Eq. 8. Since the reduction technique we considered in this paper is to keep the $\tilde{M}_{l-1}$ neurons of $\underline{x}_{l-1}$ with the largest $\tilde{M}_{l-1}$ diagonal elements of $\mathbf{C}_{l-1}$, we propose

$$J_l(\vartheta_l) = J_l(\tilde{M}_{l-1}) = \frac{\sum_{i=\tilde{M}_{l-1}+1}^{M_{l-1}} \sigma_i^2}{\sum_{i=1}^{\tilde{M}_{l-1}} \sigma_i^2}, \tag{14}$$

where $\sigma_1^2 \geq ... \geq \sigma_{M_l-1}^2 \geq 0$ are the diagonal elements of $\mathbf{C}_{l-1}$ in a decreasing order.

In Eq. 13 and 14, normalized squared reduction errors are used since different layers have different numbers of neurons and also different magnitudes of weights and signals. The final optimization problem for the optimum allocation of layer reduction factors is

$$\min_{\underline{\vartheta}} J(\underline{\vartheta}) \quad s.t. \ \alpha_{tot}(\underline{\vartheta}) \geq \alpha_{tot,min}. \tag{15}$$

The total reduction factor $\alpha_{tot}(\underline{\vartheta})$ is given in Eq. 7 for pruning and in Eq. 11 for low-rank approximation. In the former case, $\alpha_{tot}(\underline{\vartheta})$ is quadratic in $\tilde{M}_{l-1}$. In the latter case, $\alpha_{tot}(\underline{\vartheta})$ is linear in k_l.

3.4. Solving the optimization problem

Eq. 15 is an integer program and hard to solve. Fig. 1 shows the cost function $J(\underline{\vartheta})$ and the feasible region $\{\underline{\vartheta} | \alpha_{tot}(\underline{\vartheta}) \geq \alpha_{tot,min}\}$ in case of pruning of a network with $L = 2$, $M_0 = 10$, $M_1 = 5$, $M_2 = 5$ and $\alpha_{tot,min} = 0.5$. The blue dots mark the integer values of $\tilde{M}_0$ and $\tilde{M}_1$ outside the feasible region and the green dots those inside.

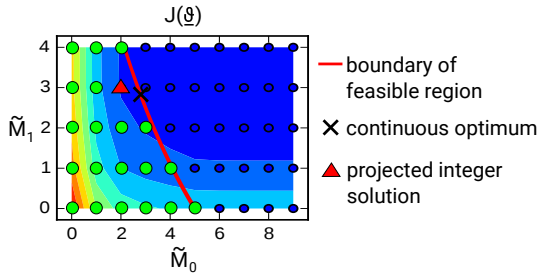

Fig. 1: The optimization problem in Eq. 15 for a network with $L = 2$, $M_0 = 10$ and $M_1 = M_2 = 5$.

For simplicity, we relax the integer program to a continuous one. We use linear interpolation of $J(\underline{\vartheta})$ at the integer grid to obtain $J(\underline{\vartheta})$ for $\underline{\vartheta} \in \mathbb{R}^L$. Its contour plot is shown in Fig. 1. Then, we solve this constrained and non-convex optimization by using the gradient-free evolutionary optimization algorithm (from the pyswarm package [10]) and project the continuous optimum to the nearest integer neighbour, see Fig. 1.

4. EXPERIMENTS

We evaluate our method on the MNIST, SVHN and NORB datasets [11, 12, 13] for different pruning factors α_{tot}. All experiments are done using Theano [14] and Keras [15]. The first two tasks are digit recognition and the last task deals with image classification. We use a full DNN with $L = 6$ layers of weights for all 3 tasks. The 5 hidden layers of neurons have the same number of 2500,2000,1500,1000 and 500 hidden

neurons for all 3 tasks, while the width of the layers changes for each task depending on the number of input features and output classes ($M_0 = 784/3072/9216$ and $M_L = 10/10/5$ for MNIST/SVHN/NORB). For all layers except for the last one, we used the rectifier linear unit as the activation function of the hidden layers and a softmax output layer. We trained the networks using rmsprop as optimizer using a learning rate of 0.003.

The full networks are trained on subsets of the datasets. Then, they are reduced with two different reduction methods, pruning and low-rank approximation as described in section 2, and two different methods for allocation of layer reduction factors, the uniform and the optimal one from section 3. The accuracy of the three networks after model reduction is evaluated by a holdout test set.

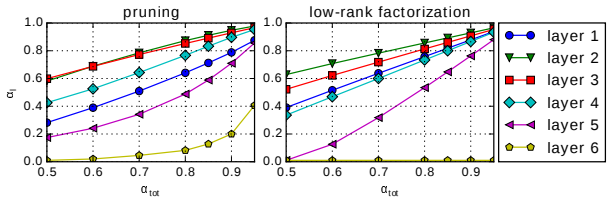

Fig. 2: Optimum layer reduction factors vs total reduction factor for a MNIST network with $L = 6$ layers.

Fig. 2 shows the optimal layer reduction factors α_l for the MNIST network for a varying total reduction factor α_{tot}. Except for the first layer $l = 1$, obviously, the top layers have much larger reduction factors, indicating a higher redundancy in the top layers. This is not surprising because the top layers have larger numbers of neurons. The output layer is not reduced at all in case of low-rank approximation. This is meaningfull, since the weight matrix of the last layer already has low-rank ($K_L = 10/10/5$ for MNIST/SVHN/NORB) and approximation with fewer singular vectors will result in large errors.

The accuracy of the reduced DNNs is shown in Fig. 3. In case of pruning, the optimum allocation of α_l greatly improves the performance of the reduced networks over a uniform reduction factor for the same total reduction factor α_{tot}. Motivated by the results in Fig. 2, that in case of low-rank approximation all α_l, $1 \leq l \leq L-1$ converge to a uniform factor $\alpha_1 = ... = \alpha_{L-1} = \alpha_{tot}$, we choose the uniform layer reduction factors for low-rank approximation $\alpha_1, ...\alpha_{L-1} = \frac{\alpha\left(\sum_{l=1}^{L} M_{l-1}M_l\right) - M_{L-1}M_L}{\sum_{l=1}^{L-1} M_{l-1}M_l}$ and $\alpha_L = 0$, i.e. the last layer is not reduced. In this case, there is almost no accuracy difference for uniform and optimal selection of α_l, since uniform selection is already close to the optimum for large α_{tot}. Another advantage of low-rank approximation is that it does not require any additional input data. The knowledge of the weight matrices $\mathbf{W}_l$ from the training is sufficient.

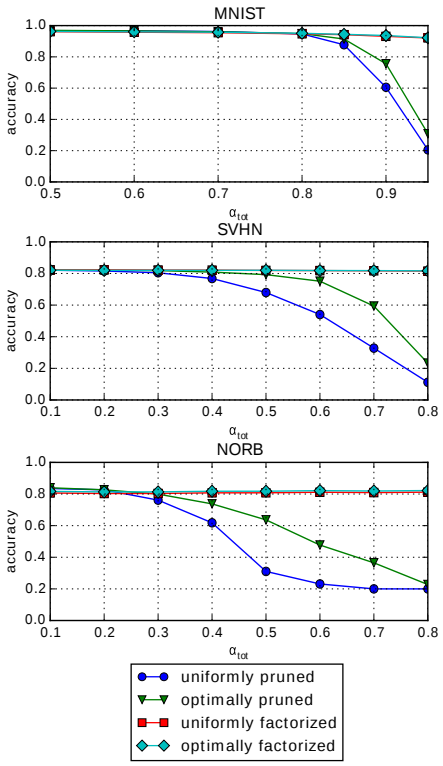

Fig. 3: Accuracy of the reduced MNIST, SVHN and NORB network vs the total reduction factor α_{tot} using the uniform and optimal layer reduction factors.

On the other hand, the naive pruning used in this paper is heuristic and far from optimum. It keeps only those neurons with high power (diagonal elements of $\mathbf{C}_{l-1}$) and does not take their correlation (off-diagonal elements in $\mathbf{C}_{l-1}$) into account. If some input neurons in a layer are highly correlated, part of them can be pruned even if they have high powers because we will be able to recover the output of this layer by using the remaining input neurons and some adapted weights and biases. In other words, there is still a big room for improvement if we use more advanced pruning methods instead of naive pruning. This is currently under study.

5. CONCLUSION

We studied how to allocate layer reduction factors for the model reduction of a trained fully connected DNN. We showed that there is a performance difference between uniform and non-uniform allocation. In particular, we presented a novel method for optimal allocation of layer reduction factors in combination with pruning and low-rank approximation. Experiments on 3 benchmark datasets validated the superior performance of this method. We conclude that it is mandatory to carefully choose the layer reduction factors.

References

[1] Li Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, 2014.

[2] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent, "Unsupervised feature learning and deep learning: A review and new perspectives," *CoRR*, vol. abs/1206.5538, 2012.

[3] Yoshua Bengio, "Practical recommendations for gradient-based training of deep architectures," *CoRR*, vol. abs/1206.5533, 2012.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., pp. 1097–1105. Curran Associates, Inc., 2012.

[5] Song Han, Jeff Pool, John Tran, and William J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, vol. abs/1506.02626, 2015.

[6] Yann Le Cun, John S. Denker, and Sara A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*. 1990, pp. 598–605, Morgan Kaufmann.

[7] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6655–6659.

[8] Adam Polyak and Lior Wolf, "Channel-level acceleration of deep face representations.," *IEEE Access*, vol. 3, pp. 2163–2175, 2015.

[9] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *CoRR*, vol. abs/1607.03250, 2016.

[10] Ahmed A. Esmin, Rodrigo A. Coelho, and Stan Matwin, "A review on particle swarm optimization algorithm and its variants to clustering high-dimensional data," *Artif. Intell. Rev.*, vol. 44, no. 1, pp. 23–45, June 2015.

[11] Li Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, , no. 141-142, November 2012.

[12] Yann LeCun, Fu Jie Huang, and Léon Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA, 2004, CVPR'04, pp. 97–104, IEEE Computer Society.

[13] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[14] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010, Oral Presentation.

[15] François Chollet, "Keras," 2015.