A NOVEL LAYERWISE PRUNING METHOD FOR MODEL REDUCTION OF FULLY CONNECTED DEEP NEURAL NETWORKS

Lukas Mauch and Bin Yang

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

ABSTRACT

Deep neural networks (DNN) are powerful models for many pattern recognition tasks, yet they tend to have many layers and many neurons resulting in a high computational complexity. This limits their application to high-performance computing platforms. In order to evaluate a trained DNN on a lower-performance computing platform like a mobile or embedded device, model reduction techniques which shrink the network size and reduce the number of parameters without considerable performance degradation performance are highly desirable. In this paper, we start with a trained fully connected DNN and show how to reduce the network complexity by a novel layerwise pruning method. We show that if some neurons are pruned and the remaining parameters (weights and biases) are adapted correspondingly to correct the errors introduced by pruning, the model reduction can be done almost without performance loss. The main contribution of our pruning method is a closed-form solution that only makes use of the first and second order moments of the layer outputs and, therefore, only needs unlabeled data. Using three benchmark datasets, we compare our pruning method with the low-rank approximation approach.

***Index Terms*—** Deep neural networks, model reduction, pruning, parameter adaptation

1. INTRODUCTION

Deep neural networks (DNN) are the state of the art method for many machine learning tasks such as image recognition, segmentation and natural language processing [1, 2]. However, evaluating a trained DNN is computationally demanding if the network is very deep and consists of layers with many neurons. Therefore, DNNs are limited to applications with enough computational power and memory. They are not suitable for mobile and embedded devices.

Both the ability of a DNN to learn an arbitrary nonlinear relationship and its computational complexity depend on the depth of the network and the width of its layers [3, 4]. A DNN with many parameters allows for learning an arbitrary complex transfer function, but is computationally expensive. Using a shallow DNN with a few narrow layers reduces the computational complexity, but also limits the achievable complexity of the transfer function [5]. There is no theory today how to choose the most efficient DNN structure, i.e. a DNN with a minimum number of layers and neurons while allowing for a transfer function that is complex enough to solve a given task.

In practice, very deep networks with a huge number of parameters are favoured because they can solve a broad class of tasks with high accuracy [6, 7]. However, most likely the chosen DNN structure has much more parameters than needed and a model with less parameters could solve the task with the same accuracy. To obtain an efficient DNN, we could optimize the DNN structure during training, e.g. by varying the number of layers and neurons. But this leads to a difficult combinatorial optimization problem. Another possibility is to start with a trained model with many parameters and try to reduce the number of parameters after training without reducing the accuracy. This technique is called model reduction.

There are two different approaches for model reduction. Some previous works used low-rank approximations of the weight matrices [8]. We call this the factorization approach. Other works reduced the number of parameters by removing (pruning) some neurons [9, 10]. This approach is called pruning. These works focused on how to select the neurons to be pruned since this selection is also a combinatorial optimization problem and therefore hard to solve. But they did not discuss how to adapt the remaining parameters after pruning.

There are three different setups for model reduction: a) We only know the trained model but have no training data. Only structures in the weight matrices can be used for model reduction. All model reduction methods described above fall into this category. b) We know the trained model and have additional unlabeled data. Therefore, we can exploit the correlations in the network outputs to reduce the network. Both (factorization and pruning) approaches can also be applied here. However, there are currently no investigations in this direction. c) We know the trained model and have labeled training data. In this case, methods from a) or b) can be used for initialization of the network and supervised fine-tuning with the labeled training data can be used to further increase the accuracy of the network.

This paper deals with case b). The idea is that some of the neurons with strongly correlated activations can be removed. By exploiting these correlations, the weights and biases for

the remaining neurons can be adapted in order to get nearly the same transfer function of the network. In this work, we derive a closed-form solution to this problem. To do so, we solve the pruning problem for each layer separately and use the least squares (LS) criterion to minimize the difference between the activation of the pruned and the full layers.

2. PRUNING OF NETWORK

2.1. Fully connected DNN

We consider pruning of fully connected DNNs as shown in Fig. 1. The transfer function $\underline{x}_L = f(\underline{x}_0, \underline{\theta})$ of the DNN consisting of L layers of weights [1] is defined by a sequence of nonlinear transformations

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

with the activation $\underline{a}_l \in \mathbb{R}^{M_l}$, the weight matrix $\mathbf{W}_l \in \mathbb{R}^{M_l \times M_{l-1}}$, the bias vector $\underline{b}_l \in \mathbb{R}^{M_l}$ and the nonlinear activation function $\Phi_l(\cdot)$ which is applied elementwise to $\underline{a}_l$. Layer l has M_l neurons. The parameter vector of the DNN is $\underline{\theta}$ and contains all elements of $\mathbf{W}_l$ and $\underline{b}_l$, $1 \le l \le L$.

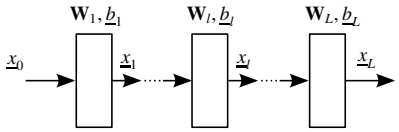

Fig. 1. Parametrization of a fully connected DNN

In supervised learning, the parameter vector $\underline{\theta}$ is estimated by minimizing a cost function

$$\hat{\underline{\theta}} = \arg\min_{\underline{\theta}} J(\{\underline{x}_0(n)\}, \{\underline{y}(n)\}, \underline{\theta}), \tag{3}$$

where $\{\underline{y}(n)\}$, $1 \le n \le N$ is a sequence of reference samples $\underline{y}(n) \in \mathbb{R}^{M_L}$ for the network output. For DNN-based classification, $\{\underline{y}(n)\}$ contains the correct class labels for a given input sequence $\{\underline{x}_0(n)\}$ and the negative log-likelihood or the cross-entropy can be used for $J(\{\underline{x}_0(n)\}, \{\underline{y}(n)\}, \underline{\theta})$ [4]. For DNN-based regression, $\underline{y}(n)$ represents the desired signal and the LS criterion is often used. Below we refer to such a trained fully connected network as the "full" network and derive methods to reduce the number of parameters in $\hat{\underline{\theta}}$ without considerable performance degradation.

2.2. Problem formulation

Pruning a trained network means to delete some of its neurons. This is visualized in Fig. 2 for layer l. The layer of the full network with parameters $\mathbf{W}_l$ and $\underline{b}_l$ is shown on the left, while the pruned layer is given on the right. Some elements of the input vector $\underline{x}_{l-1}$ of layer l (i.e. the corresponding neurons in layer $l-1$) are deleted and do not contribute to the activation $\tilde{\underline{a}}_l$ on the right. The transfer function of the pruned network becomes

$$\tilde{\underline{x}}_l = \Phi_l\left(\tilde{\underline{a}}_l\right), \quad 1 \le l \le L \tag{4}$$
$$\tilde{\underline{a}}_l = \tilde{\mathbf{W}}_l \mathbf{P}_l \tilde{\underline{x}}_{l-1} + \tilde{\underline{b}}_l, \tag{5}$$

where $\mathbf{P}_l \in \mathbb{R}^{M_{l-1}^p \times M_{l-1}}$ $(M_{l-1}^p < M_{l-1})$ is a selection matrix selecting M_{l-1}^p elements from $\tilde{\underline{x}}_{l-1} \in \mathbb{R}^{M_{l-1}}$. Each row of $\mathbf{P}_l$ has zero elements but one entry equal to one and all rows of $\mathbf{P}_l$ are orthogonal. The weight matrix of the pruned layer $\tilde{\mathbf{W}}_l \in \mathbb{R}^{M_l \times M_{l-1}^p}$ has only M_{l-1}^p columns. Therefore, we only need $M_l M_{l-1}^p$ multiplications instead of $M_l M_{l-1}$.

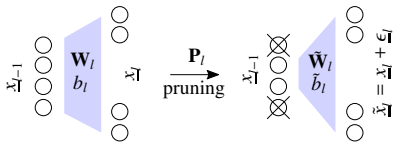

Fig. 2. Pruning of a fully connected network layer

A naive approach to select $\tilde{\mathbf{W}}_l$ would be

$$\tilde{\mathbf{W}}_l = \mathbf{W}_l \mathbf{P}_l^T. \tag{6}$$

This means, we simply delete all columns in $\mathbf{W}_l$ corresponding to the pruned elements in $\tilde{\underline{x}}_{l-1}$ without changing the other elements of $\mathbf{W}_l$. This is done in all pruning and dropout methods up to now. This naive pruning introduces the error

$$\underline{\epsilon}_l = \tilde{\underline{a}}_l - \underline{a}_l \tag{7}$$

to the activation of the pruned layer and the DNN performance will most likely decrease. In order to get nearly the same performance as with the full network, we have to adjust $\tilde{\mathbf{W}}_l$ and $\tilde{\underline{b}}_l$ such to minimize the mean squared error (MSE)

$$\mathrm{MSE}_l = \mathrm{E}\left[\|\underline{\epsilon}_l\|^2\right] \tag{8}$$

in each layer. The intuition tells us that if the pruned neurons are correlated with some remaining neurons in the same layer, we should be able to use the remaining neurons and adapted $\tilde{\mathbf{W}}_l$ and $\tilde{\underline{b}}_l$ to minimize the difference between $\tilde{\underline{a}}_l$ and $\underline{a}_l$.

We use a set of N unlabeled training samples $\mathcal{X} = \{\underline{x}_0(1), \ldots, \underline{x}_0(N)\}$ to calculate MSE_l. Then we minimize

$$\sum_{n=1}^{N} \|\tilde{\underline{a}}_l(n) - \underline{a}_l(n)\|^2 = \sum_{n=1}^{N} \|\tilde{\mathbf{W}}_l \mathbf{P}_l \tilde{\underline{x}}_{l-1}(n) + \tilde{\underline{b}}_l - \mathbf{W}_l \underline{x}_{l-1}(n) - \underline{b}_l\|^2 \tag{9}$$

over $\tilde{\mathbf{W}}_l$ and $\tilde{\underline{b}}_l$. Here, we make two assumptions to simplify the model reduction problem. First, we assume that $\mathbf{P}_l$ is given and independent of $\tilde{\mathbf{W}}_l$ and $\tilde{\underline{b}}_l$. Second, we do not adjust $\tilde{\mathbf{W}}_l$ and $\tilde{\underline{b}}_l$ of different layers jointly, as this would result in a nonlinear LS problem due to the nonlinear activation

[1] This corresponds to $L-1$ hidden layers of neurons.

functions Φ_l. Instead, we adjust $\tilde{\mathbf{W}}_l$ and $\underline{\tilde{b}}_l$ layerwise by assuming $\underline{\tilde{x}}_{l-1} = \underline{x}_{l-1}$ in Eq. 9, starting with the first layer $l = 1$. The cost function in Eq. 9 then simplifies to

$$\min_{\tilde{\mathbf{W}}_l, \underline{\tilde{b}}_l} \sum_{n=1}^{N} \|(\tilde{\mathbf{W}}_l \mathbf{P}_l - \mathbf{W}_l)\underline{x}_{l-1}(n) + \underline{\tilde{b}}_l - \underline{b}_l\|^2. \quad (10)$$

2.3. Closed-form solution

Setting the derivative of the cost function in Eq. 10 with respect to $\underline{\tilde{b}}_l$ to zero leads to

$$\underline{\tilde{b}}_l = \underline{b}_l + (\mathbf{W}_l - \tilde{\mathbf{W}}_l \mathbf{P}_l)\underline{\mu}_{l-1}, \quad (11)$$

where

$$\underline{\mu}_{l-1} = \frac{1}{N} \sum_{n=1}^{N} \underline{x}_{l-1}(n) \quad (12)$$

is the sample mean of the input of full layer l. This means, $\underline{\tilde{b}}_l$ is obtained by adding a correction term to $\underline{b}_l$ which is equal to the difference between the mean output of the full layer l and the pruned layer l. Inserting Eq. 11 into Eq. 10 further simplifies the cost function to

$$\sum_{n=1}^{N} \|(\tilde{\mathbf{W}}_l \mathbf{P}_l - \mathbf{W}_l)(\underline{x}_{l-1}(n) - \underline{\mu}_{l-1})\|^2. \quad (13)$$

The solution $\tilde{\mathbf{W}}_l$ minimizing it is given by

$$\tilde{\mathbf{W}}_l = \mathbf{W}_l \mathbf{C}_{l-1} \mathbf{P}_l^T \left[\mathbf{P}_l \mathbf{C}_{l-1} \mathbf{P}_l^T\right]^{-1}, \quad (14)$$

where

$$\mathbf{C}_{l-1} = \frac{1}{N} \sum_{n=1}^{N} (\underline{x}_{l-1}(n) - \underline{\mu}_{l-1})(\underline{x}_{l-1}(n) - \underline{\mu}_{l-1})^T \quad (15)$$

is the sample covariance matrix of $\underline{x}_{l-1}$.

If $\mathbf{C}_{l-1}$ is the identity matrix, i.e. all elements of $\underline{x}_{l-1}$ are uncorrelated with equal variance, Eq. 14 reduces to $\tilde{\mathbf{W}}_l = \mathbf{W}_l \mathbf{P}_l^T$, the naive selection in Eq. 6. In this case, the information loss due to pruning of some elements in $\underline{x}_{l-1}$ can not be recovered from the other uncorrelated elements of $\underline{x}_{l-1}$ and the best we can do for $\mathbf{W}_l$ is doing nothing. If, however, the elements of $\underline{x}_{l-1}$ are correlated, the pruning of some elements of $\underline{x}_{l-1}$ makes an adaption of $\mathbf{W}_l$ and $\underline{b}_l$ necessary in order to recover the activation $\underline{a}_l$ of layer l as good as possible.

2.4. Selection strategy

Let $\alpha_l \in (0, 1)$ be the reduction factor for layer l, i.e. we delete round$(\alpha_l M_{l-1})$ elements of $\underline{x}_{l-1}$ and $M_{l-1}^p = M_{l-1} -$ round$(\alpha_l M_{l-1})$. For simplicity, we consider a constant reduction factor $\alpha_l = \alpha$ for all layers. The number of multiplications for layer l is then reduced from $M_l M_{l-1}$ to $M_l^p M_{l-1}$.

The best way to select $\mathbf{P}_l$ is to minimize MSE$_l$ in Eq. 8. However, optimization with respect to $\mathbf{P}_l$ is a combinatorial optimization problem which is hard to solve. Due to limited space, we are not going to discuss different suboptimal selection strategies and their impact to model reduction. This will be done in a future publication. This paper considers the two easiest selection strategies: a) select round(αM_{l-1}) elements of $\underline{x}_{l-1}$ to be pruned at random for a given reduction factor α like dropout [11], b) choose $\mathbf{P}_l$ to minimize Eq.10 for the naive case (naive selection), i.e. we keep all elements of $\underline{x}_{l-1}$ with the largest diagonal elements in $\mathbf{C}_{l-1}$. In both cases, the selection is done independently for all layers.

2.5. Network reduction with low rank approximation

We compare our pruning method with model reduction based on low-rank approximation of the weight matrices. First, each $\mathbf{W}_l$ of the trained DNN undergoes a singular value decomposition (SVD)

$$\mathbf{W}_l = \mathbf{U}_l \mathbf{\Sigma}_l \mathbf{V}_l^T \in \mathbb{R}^{M_l \times M_{l-1}}. \quad (16)$$

Then we approximate $\mathbf{W}_l$ by a low-rank matrix $\tilde{\mathbf{W}}_l = \mathbf{U}_{l,K_l} \mathbf{\Sigma}_{l,K_l} \mathbf{V}_{l,K_l}^T$, where $\mathbf{U}_{l,K_l} \in \mathbb{R}^{M_l \times K_l}$ and $\mathbf{V}_{l,K_l} \in \mathbb{R}^{M_{l-1} \times K_l}$ are matrices containing the K_l left and right singular vectors belonging to the K_l largest singular values $\lambda_1, ..., \lambda_{K_l}$ of $\mathbf{W}_l$ and $\mathbf{\Sigma}_{l,K_l} = \text{diag}(\lambda_1, ..., \lambda_{K_l})$. The number of multiplications is then reduced from $M_{l-1} M_l$ for $\mathbf{W}_l \underline{x}_{l-1}$ to $K_l M_{l-1} + K_l + K_l M_l$ for $\mathbf{U}_{l,K_l} \mathbf{\Sigma}_{l,K_l} \mathbf{V}_{l,K_l}^T \underline{x}_{l-1}$. In order to achieve the same reduction factor α as previously, we choose

$$K_l = \text{round}\left((1 - \alpha) \frac{M_{l-1} M_l}{M_{l-1} + M_l + 1}\right). \quad (17)$$

We do not reduce the last layer ($K_L = \min(M_L, M_{L-1})$), since it has already low rank and reducing it will result in large reduction errors.

3. EXPERIMENTS

We evaluate our pruning method on the MNIST, SVHN and NORB datasets [12, 13, 14] for different reduction factors α and compare it to the low-rank approximation. All experiments are done using Theano [15] and Keras [16]. The first two tasks are digit recognition and the last task deals with image classification. The goal of these experiments is not to present the best DNN for a particular task, but rather to compare the classification performance between the full DNN and that after pruning or low-rank approximation. Hence, the used full DNN is not the most sophisticated one.

For simplicity, 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 a fraction of the dataset. They are then reduced and their accuracy is compared to the full networks using a holdout testset. We compare four methods of model reduction: a) low-rank approximation as described in section 2.5, b) naive pruning assuming $\mathbf{C}_{l-1} = \mathbf{I}$, c) our pruning method by exploiting the correlations of the neuron outputs, using random selection of $\mathbf{P}_l$, d) our pruning method, using the naive selection of $\mathbf{P}_l$.

Fig. 3 shows the classification accuracy of the 3 tasks for both the full DNN and the 4 pruned networks for a reduction factor $0.1 \leq \alpha \leq 0.8$. We first compare the performance of the 3 full networks. Clearly, the MNIST dataset receives the highest accuracy of 97% while the SVHN and NORB dataset achieve an accuracy of 82% and 84%, respectively. This is easy to understand because MNIST contains clean, portrait oriented grayscale digits while SVHN contains noisy, arbitrary oriented coloured digits. Since the 3 full DNNs have the same depth and layer widths (except for the input and output layer), the full DNN for MNIST is almost optimum while the other two full DNNs can be further improved. Nevertheless, all 3 full DNNs are over dimensioned and can be shrinked by model reduction techniques.

Now we compare the performance of the full DNNs and their pruned ones. Though the effect of model reduction varies from task to task, there are some common observations in Fig. 3. Naive pruning, i.e. delete some neurons in each layer without weight adaptation, degrades the network performance early. In the simple SVHN task, a reduction factor of 40% reduced the network accuracy by more than 10% and a reduction factor of 55% achieved only an accuracy of about 60%. For NORB, the performance degradation is even more serious. The reason is that naive pruning changes the activation of the remaining layers.

In contrast, our pruning method with a layerwise weight and bias adaptation shows excellent results in all 3 tasks. In the MNIST task, there is almost no performance degradation, even for a pruning factor of 80%. This demonstrates that the full DNN for this task is highly redundant and overdimensioned. For the other two more challenging tasks, there is a certain accuracy degradation for high pruning factors ($\alpha \geq 50\%$ for SVHN and $\alpha \geq 70\%$ for NORB). Our pruning method shows comparable results to low-rank approximation, even surpassing it in case of MNIST, meaning that using weight adaptation is mandatory for a good network reduction. Although, there still remains a small performance gap between pruning and low-rank approximation in the case of SVHN and NORB. This is due to the random selection strategy in section 2.4. Using the naive selection strategy already helps to close this gap in case of SVHN. Therefore, we believe that by using a better selection strategy and/or by using a fine-tuning learning with some labeled data, pruning will show better peformance than low-rank approximation in almost any case, and that the pruned network can almost achieve the original performance of the full network. This will be addressed in a future work.

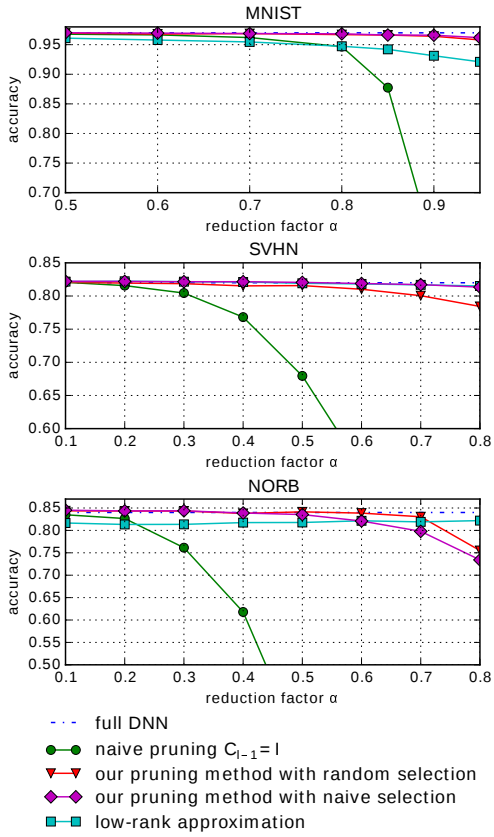

Fig. 3. Comparison of 4 model reduction methods

4. CONCLUSION

We presented a novel model reduction method for a trained fully connected DNN to reduce its computational complexity and memory consumption. It makes use of the mean and covariance matrix of the output of each layer and, hence, needs only unlabeled data. By exploiting the correlations of neurons in a layer, we derived a closed-form solution for the adaptation of the weights and biases of the remaining neurons in order to recover the original activations as good as possible. In three classification experiments, our method significantly outperforms the low-rank approximation and shows the same or a comparable accuracy as the full network for a reduction factor up to 70%.

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] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio, "On the number of linear regions of deep neural networks," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., pp. 2924–2932. Curran Associates, Inc., 2014.

[4] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[5] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015, Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

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

[7] 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.

[8] 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.

[9] Song Han, Jeff Pool, John Tran, and William Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., pp. 1135–1143. Curran Associates, Inc., 2015.

[10] 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.

[11] Pierre Baldi and Peter J Sadowski, "Understanding dropout," in *Advances in Neural Information Processing Systems 26*, C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, Eds., pp. 2814–2822. Curran Associates, Inc., 2013.

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

[13] 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.

[14] 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.

[15] 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.

[16] François Chollet, "Keras," 2015.