Convolutional Neural Networks with analytically determined Filters

1st Matthias Kissel
Chair of Data Processing
Technical University of Munich
80333 Munich, Germany
matthias.kissel@tum.de

2nd Klaus Diepold
Chair of Data Processing
Technical University of Munich
80333 Munich, Germany
kldi@tum.de

2022 International Joint Conference on Neural Networks (IJCNN) | 978-1-7281-8671-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/IJCNN55064.2022.9891906

***Abstract*—In this paper, we propose a new training algorithm for Convolutional Neural Networks (CNNs) based on well-known training methods for neural networks with random weights. Our algorithm analytically determines the filters of the convolutional layers by solving a least squares problem using the Moore-Penrose generalized inverse. The resulting algorithm does not suffer from convergence issues and the training time is drastically reduced compared to traditional CNN training using gradient descent. We validate our algorithm with several standard datasets (MNIST, FashionMNIST and CIFAR10) and show that CNNs trained with our method outperform previous approaches with random or unsupervisedly learned filters in terms of test prediction accuracy. Moreover, our approach is up to 25 times faster than training CNNs with equivalent architecture using a gradient-descent based algorithm.**

***Index Terms*—Convolutional Neural Network, Pseudo-Inverse, Gradient-Free Training, Efficient Training**

I. Introduction

In recent years, Deep Neural Networks (DNNs) have been able to solve more and more complex tasks by becoming larger and deeper [6] [22]. Typically, DNNs are trained using gradient descent methods. Besides achieving remarkable results, this can, however, result in long training times. Depending on the available hardware, a training session can easily last for several days. Moreover, the selection of hyper parameters for the training can play a crucial role for the training results. This includes for example momentum terms or the step size of the gradient descent method.

One approach to overcome these problems is to use neural networks with random weights, whereas only the output layer of the network is trained. This approach has been rediscovered several times and was published under different names [1]. Early work on this subject dates back to 1992 [18], [20]. This paper is based on the training algorithm proposed from Huang et al. [8], which is called Extreme Learning Machine (ELM). They propose to only train the last layer of the neural network, i.e. the output layer. All other weights and biases of the network are initialized randomly and not altered at all. Since only the last layer is trained, the parameters can be determined analytically using the Moore-Penrose generalized inverse. They showed that their training scheme drastically reduces the training time while achieving competitive results.

Since CNNs are the state of the art in image recognition [10], several approaches have been proposed in order to combine CNNs with the training scheme for neural networks with random weights [5], [7], [16]. One of the main drivers of this is that CNNs tend to have very deep architectures [6], which can result in long training times.

In this paper, we extend a training algorithm for two-hidden-layer ELMs proposed from Qu et al. [19] to be used for multi-hidden layered CNNs. By that, the parameters in the convolutional layers are determined analytically. In particular, no gradient descent method is used for the training of the (potentially deep) CNN. Instead, the weights are determined analytically using Moore-Penrose generalized inverses. This results in a robust and very fast training algorithm for CNNs. We show that our approach outperforms existing approaches for CNNs with random weights as well as unsupervisedly learned filters on several standard image classification data sets.

The rest of the paper is organized as follows. We first give an overview over existing approaches of combining CNNs with the training scheme for neural networks with random weights. In the following section, we introduce our proposed training algorithm based on the existing theory for neural networks with random weights. We validate our approach with several standard image recognition data sets in the subsequent section. Finally, we summarize our findings and give a conclusion.

II. Literature Review

Using neural networks with random weights has been proposed several times in literature [1], [8], [9], [14], [17], [18], [20]. The common approach for neural networks with random weights is to initialize the weights in the hidden layers randomly and not altering them at all during the training phase. Solely the output weights of the network are trained, often by analytically determining these weights. This training scheme potentially overcomes some common problems observed when training neural networks [9]. These problems include long training times using gradient descent based training algorithms, hyper parameter selection such as learning rates, and convergence problems during training. In this paper, we focus on the work in the area of neural networks with random weights. However, we would like to point out that

there are also other approaches for training neural networks without using the famous backpropagation algorithm [2], [11], [13], [21].

CNNs are nowadays the state of the art in image recognition [10]. In order to combine the advantages of CNNs and the training scheme for random neural networks, different approaches have been developed [4], [5], [7], [16]. Huang et al. [7] argued that following from existing ELM theory, different types of local receptive fields can be used in random neural networks. They proposed a Local Receptive Field based Extreme Learning Machine (LRF-ELM) analogous to a CNN architecture. Based on the LRF-ELM, Pang et al. [16] proposed a deep Convolutional Extreme Learning Machine (CELM), which they evaluated on two handwritten digit data sets. Ding et al. [4] proposed a CELM with kernels (CKELM) and Dos Santos et al. [5] proposed a deep CELM, which combines different types of convolutional filters.

There are different ways to determine the filters in the convolutional layers. For example, they can be generated randomly based on a probability distribution [5], [7], [16]. Other approaches use randomly extracted patches from the training images as filters [5], [15] or learn filters with an unsupervised learning method, for example based on Principle Component Analysis [3] or k-means [23]. Furthermore, filters can be generated according to mathematical models (e.g. of edge detectors or gabor filters) [5], [15] or transferred from a trained model of another data set [5], [15].

To the best of our knowledge, it has not yet been proposed to determine the filters in CNNs analytically in the same way as the output weights in neural networks with random weights.

III. Method

Our training algorithm is based on the ELM training scheme introduced by Huang et al. [8], [9] and the training algorithm proposed from Qu et al. [19] for two-hidden-layer ELMs. Therefore, we first recapitulate the existing algorithms and subsequently present our training algorithm for CNNs.

A. Extreme Learning Machine

ELMs [8], [9] are single hidden layer feedforward neural networks of the form

$$F(\boldsymbol{x}, W_1, W_2, \boldsymbol{\theta}) = \sigma(W_1 \boldsymbol{x} + \boldsymbol{\theta})^T W_2, \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^m$ are inputs to the neural network, $W_1 \in \mathbb{R}^{z \times m}$ and $W_2 \in \mathbb{R}^{z \times k}$ are the weight matrices of the hidden and the output layer respectively, $\boldsymbol{\theta} \in \mathbb{R}^z$ is the bias vector of the hidden layer and σ is a nonlinear activation function, which is applied element-wise to its inputs. The output activation function is chosen to be linear.

Our aim is to chose the parameters W_1, W_2 and $\boldsymbol{\theta}$ such that $F(\boldsymbol{x}, W_1, W_2, \boldsymbol{\theta})$ approximates a desired function $\hat{F}(\boldsymbol{x})$. For that, we have n arbitrary distinct samples $(\boldsymbol{x_i}, \boldsymbol{t_i})$ with $\boldsymbol{x_i} \in \mathbb{R}^m$ and $\boldsymbol{t_i} \in \mathbb{R}^k$ such that $\hat{F}(\boldsymbol{x_i}) = \boldsymbol{t_i}$ for $i = 1, ..., n$, which are concatenated into the training input matrix $X = [\boldsymbol{x_1} \dots \boldsymbol{x_n}]^T \in \mathbb{R}^{n \times m}$ and training output matrix $T = [\boldsymbol{t_1} \dots \boldsymbol{t_n}]^T \in \mathbb{R}^{n \times k}$.

In contrast to the standard approach of learning the parameters W_1, W_2 and $\boldsymbol{\theta}$ using a training algorithm based on gradient-descent, in ELMs, W_1 and $\boldsymbol{\theta}$ are initialized randomly and not trained at all. The output weights W_2 can be determined analytically using the Moore-Penrose generalized inverse. For that, the training data samples are propagated through the network to obtain the hidden layer output matrix

$$H = \begin{bmatrix} \sigma(W_1 \boldsymbol{x_1} + \boldsymbol{\theta}) & \dots & \sigma(W_1 \boldsymbol{x_n} + \boldsymbol{\theta}) \end{bmatrix}^T \in \mathbb{R}^{n \times z}, \tag{2}$$

where z is the number of neurons in the hidden layer. Thus we can solve the problem

$$min_{W_2} ||F(\boldsymbol{x}, W_1, W_2, \boldsymbol{\theta}) - T||^2 = min_{W_2} ||HW_2 - T||^2 \tag{3}$$

using the Moore-Penrose generalized inverse. Hence, the weights are determined by

$$W_2 = H^{\dagger} T, \tag{4}$$

where $\dagger$ denotes the Moore-Penrose generalized inverse. The obtained solution is unique and minimizes the norm of the weights obtained besides minimizing the training error [9].

In order to increase the robustness against small singular values in the hidden layer output matrix, regularization terms can be added to the least squares problem. Throughout this paper, we use ridge regression for solving the least squares problem [12]. For that, we add a regularization term to the least-squares problem

$$min_{W_2} ||HW_2 - T||^2 + \lambda ||W_2||^2, \tag{5}$$

where λ is a hyperparameter which weighs the importance of the regularization term. Note that usually it can not be guaranteed that the hidden output layer matrix H is centered. Therefore, in order to further increase the stability of the least squares solution, we add biases to the output layer neurons $\boldsymbol{\theta_2}$. The biases are determined in order to center the data matrices, i.e. they are used as intercepts for the ridge regression problem

$$\boldsymbol{\theta_2} = av_1(T) - av_1(H)^T W_2, \tag{6}$$

where av_1 denotes the average over all elements along the first axis of the respective matrix. Of course, the ridge regression is performed with centered inputs and outputs if biases are used.

B. Two-hidden-layer Extreme Learning Machine

Qu et al. [19] proposed to extend the training scheme of ELMs to two-hidden-layer ELMs $F_{Qu}(\boldsymbol{x}, W_1, W_2, W_3, \boldsymbol{\theta_1}, \boldsymbol{\theta_2})$, where W_1 is the weight matrix connecting the inputs to the first hidden layer, W_2 comprises the weights between the first and the second hidden layer (bias vectors $\boldsymbol{\theta_1}$ and $\boldsymbol{\theta_2}$ respectively) and W_3 comprises the weights of the output layer. They proposed to obtain the parameters W_2 and $\boldsymbol{\theta_2}$ analogously to the output weights W_3 by analytically solving a linear least-squares problem. For this, the desired pre-nonlinearity activations for the hidden layer are needed, which can be calculated by

$$H_{2,des}^{pre} = \sigma^{-1}(TW_3^{\dagger}). \tag{7}$$

Accordingly, in order to determine the desired activation at the hidden layer, the activation function σ needs to be invertible. Using activation functions like hyperbolicus tangens or sigmoid, the feasibility of the inversion can be guaranteed by normalizing to specific value ranges [19] or simple value clipping. In our experiments, however, we use the popular Leaky-ReLU function. Since this activation function is defined on whole $\mathbb{R}$ and the outputs are not bounded when suitable hyperparameters are used, the inverse function is also defined and unique on whole $\mathbb{R}$. Therefore, we do not need any extra normalization or clipping steps.

When $H_{2,des}^{pre}$ is known, the parameters W_2 and $\boldsymbol{\theta_2}$ can be determined by solving the linear least-squares problem

$$min_{W_2,\boldsymbol{\theta_2}} || \begin{pmatrix} H_1 & \mathbf{1} \end{pmatrix} \begin{pmatrix} W_2 \\ \boldsymbol{\theta_2}^T \end{pmatrix} - H_{2,des}^{pre} ||^2, \tag{8}$$

where $\mathbf{1} \in \mathbb{R}^n$ is a vector containing ones.

We slightly adapt this training scheme. In our algorithm, the weights are determined by solving the linear least-squares problem

$$min_{W_i} ||H_{i-1}W_i - H_{i,des}^{pre}||^2 + \lambda ||W_i||^2. \tag{9}$$

The bias vectors $\boldsymbol{\theta_i}$ are used to center the data, i.e. they are used as intercepts for the least-squares solution (as described in Section III-A).

In the following, we consider neural networks with more than two hidden layers. For these the calculation of the desired activations can be extended straightforwardly

$$H_{i,des}^{pre} = \sigma^{-1}(H_{i+1}W_{i+1}^{\dagger}). \tag{10}$$

C. Analytically Determining Convolutional Filters

We propose to adapt the training scheme from Qu et al. [19] in order to apply it to CNNs. In the following, we describe our adaptations to the algorithm which consist of several particular reshaping operations. This results in a training algorithm, which combines the advantages of CNNs and ELMs.

In the densely connected layers investigated from Qu et al. [19], the outputs of the hidden layer are computed by a matrix-vector multiplication followed by the nonlinear activation function. In our case, however, the hidden layer might not perform a simple matrix-vector multiplication, but convolve its multi-dimensional inputs with its multi-dimensional filter. For ease of notation, we introduce our method for four-dimensional inputs $H_{i-1} \in \mathbb{R}^{n \times s \times t \times r}$ and convolutional layers with u filters $W_i \in \mathbb{R}^{l \times w \times r \times u}$, i.e. we consider a two-dimensional convolutional over r input channels and u output channels for n input images of size $s \times t$. However, with few adaptations, our method can also be applied to convolutions of other dimensions.

Since the convolution is a linear operation, it can be represented as a matrix-vector product. Usually, the convolution is expressed by reshaping the inputs and rearranging the filter weights into a sparse toeplitz matrix. In our case, however, we propose another reshaping format which suits our application. The aim is to reshape the tensors used during the convolution operation into matrices without imposing a particular structure onto the weight matrix

$$H_i^{pre,flat} = H_{i-1}^{flat} W_i^{flat}, \tag{11}$$

where $W_i^{flat} \in \mathbb{R}^{lwr \times u}$. Moreover, H_i^{pre} denotes the i^{th} pre-activation hidden layer output matrix and H_i^{flat} refers to the i^{th} reshaped hidden layer output matrix. After computing $H_i^{pre,flat}$, the activation function is applied and the result is reshaped into the standard shape H_i.

Using these reshaping operations, the filter weights can be calculated using the Moore-Penrose generalized inverse if the desired pre-nonlinearity activations $H_{i,des}^{pre}$ are known. Unfortunately, in the general case, the desired activations for convolutional layers can not be uniquely determined. Since there are overlapping receptive fields in convolutional layers (in the general case), using the rule introduced in equation 10 leads to multiple values for each entry of the desired input activation. This problem can be solved by either averaging the computed desired activations for each entry or by setting the strides in the convolutional layer to the same value as the filter size. By that, receptive fields of the convolutional layer are not overlapping anymore and the result of the minimum-norm least squares problem for calculating the desired activations is unique. In our experiments, we show that even the approach of adapting the stride, which is supposedly inferior, leads to very good results. The reshaping operation for inputs in the case that the strides equal the filter size are depicted in Figure 1. Algorithm 1 describes the resulting training algorithm for CNNs.

IV. Experiments

We validate our training algorithm by comparing the test accuracy achieved on three standard data sets compared to results achieved with existing training algorithms. The following data sets are used in our experiments:

1) MNIST data set of handwritten digits
2) FashionMNIST data set of fashion articles
3) CIFAR10 data set of images from 10 different categories

On all these data sets, we train four models and compare the prediction accuracy on a test set, which is held out until testing. The four models are

1) CNN with filters determined using ridge regression (our approach).
2) CNN with randomly initialized filters.
3) CNN with unsupervisedly learned filters. We use the centers obtained by k-means clustering, where k is the number of filters in the respective convolutional layer.
4) CNN trained by backpropagation using the Adam optimizer. The network is trained until convergence on a validation set and the network with best validation loss during training is used for computing the test accuracy.

To account for the effects of random initialization, we repeat all experiments ten times and report the mean test accuracy and the respective standard deviation over the ten runs. Moreover, in order to analyze the influence of the network architecture

Data: Input Data $X \in \mathbb{R}^{n \times s \times t \times r}$, Target Outputs $T \in \mathbb{R}^{n \times k}$, Initialized CNN F with b layers, Regularization Hyperparameter λ
Result: Trained CNN F
for $i = 1, \ldots, b$ **do**
  /* Compute the Activation for layer i, which is the hidden layer output matrix of layer $i-1$ (Equation 2) */
  if i == *1* **then**
    $H_{i-1} = X$;
  end
  else
    Get Activation H_{i-1} at layer $i-1$ in F;
  end
  /* Compute target Activation $H_{i,des}^{pre}$ by back-propagating the desired Activations through the network (Equation 10) */
  Recursively compute $H_{i,des}^{pre}$; // This involves reshaping operations
  /* Prepare matrices for solvig the least-squares system (Figure 1) */
  if L_i *is a convolutional Layer* **then**
    Crop entries which are dropped during propagation (due to stride, padding, etc.);
    Reshape the activation $H_{i-1} \rightarrow H_{i-1}^{flat}$;
    Reshape the target activation $H_{i,des}^{pre} \rightarrow H_{i,des}^{flat,pre}$;
  end
  else
    $H_{i-1}^{flat} = H_{i-1}$;
    $H_{i,des}^{flat,pre} = H_{i,des}^{pre}$;
  end
  $H_{i-1,cen}^{flat} = H_{i-1}^{flat} - av_1(H_{i-1}^{flat})$;
  $H_{i,des,cen}^{flat,pre} = H_{i,des}^{flat,pre} - av_1(H_{i,des}^{flat,pre})$;
  /* Solve the least-squares system (Equation 9) and compute the biases (Equation 6) */
  $W_i^{flat} = argmin_{W_i^{flat}} ||H_{i-1,cen}^{flat} W_i^{flat} - H_{i,des,cen}^{flat,pre}||^2 + \lambda ||W_i^{flat}||^2$;
  $\boldsymbol{\theta}_i = av_1(H_{i,des}^{flat,pre}) - av_1(H_{i-1})^T W_i^{flat}$;
  /* Prepare Matrices for updating the layer parameters */
  if L_i *is a convolutional Layer* **then**
    Reshape the weights $W_i^{flat} \rightarrow W_i$;
  end
  else
    $W_i = W_i^{flat}$;
  end
  Update the weights of L_i to W_i;
  Update the biases of L_i to $\boldsymbol{\theta}_i$;
end

Algorithm 1: Training algorithm for CNNs consisting of convolutional layers and densely connected layers. Inputs to the algorithm are the Input Data, Target Outputs, Regularization Hyperparameter and the initialized CNN. The Algorithm trains the CNN with the approach proposed in this paper. Activations are flattened between the convolutional and densely connected layers. At each layer of the network, the activations (i.e. the hidden layer output matrix of the previous layer) and the desired activations are computed. This involves several reshaping operations, since the operations performed during propagation must be represented as matrix-vector products in order to analytically determine the convolutional filters. Based on the computed matrices, the parameters of the convolutional filters are determined using ridge regression. At the end, the trained CNN is returned.

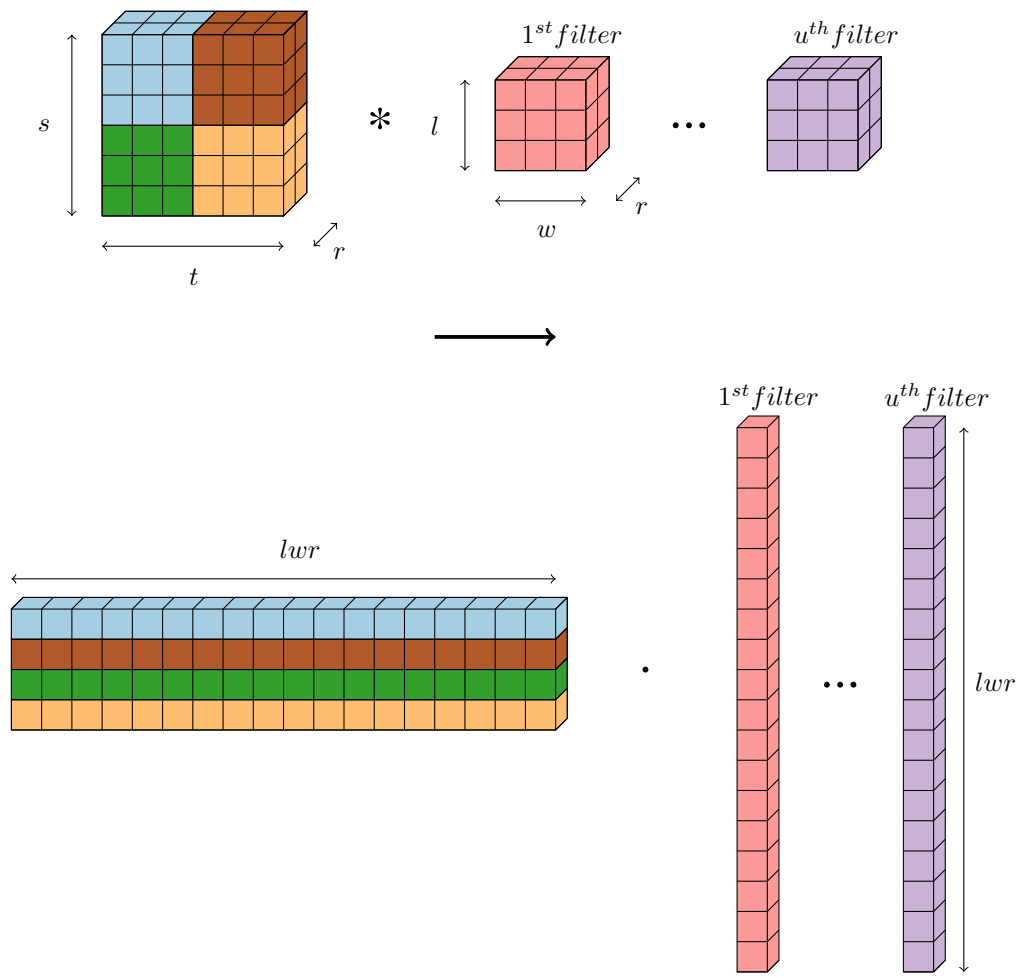

Fig. 1. Schematic Illustration of the reshaping operation. The non-overlapping receptive fields of the convolutional layer are flattened and concatenated into a two-dimensional matrix.

TABLE I
IMPORTANT HYPERPARAMETERS

Name	Value
Spatial Size of the conv. filters (l, w)	(3,3)
# filters per convolutional layer u	50
# densely connected layers	1
# neurons per densely connected layer	200
Nonlinear Activation Function	Leaky ReLU ($\alpha = 2$)
Regularization Hyperparameter λ	10^{-3}
Repetitions per Experiment	10

on the resulting performance, we compare networks with zero, one and two convolutional layers for each data set and model. For the experiments with the proposed method, the spatial size of the strides in the convolutional layers is set to match the spatial size of the convolutional filters. The spatial size of the strides for the other models is set to 1 (except for CNNs with unsupervisedly learned filters because the memory requirements were too exhaustive when setting the stride too small). We argue that this design decision disadvantages our method, since increasing the spatial size of the stride effectively reduces the number of operations performed during propagation through the neural network. Our code and all hyperparameters can be found online[1] and the most important hyper parameters are also listed in Table I. Please note that our aim is not to tweak the hyper parameters in order to beat the state-of-the art results on the investigated data sets. Instead, our aim is to provide a fair comparison of our method against standard methods to identify the strengths and weaknesses of the proposed method.

Our approach outperforms the approach of using random convolutional filters as well as unsupervisdely trained filters in all datasets for almost all architecture configurations (see Figure IV). Moreover, the improvements made by adding more layers to the architecture are higher with our approach than with the other approaches which are not based on gradient-descent. However, this effect becomes smaller for deeper networks. Since the proposed approach outperforms the non-gradient-descent methods in all datasets, we also conclude that our method is quite robust.

[1]https://github.com/MatthiasKi/analyticalconvfilters

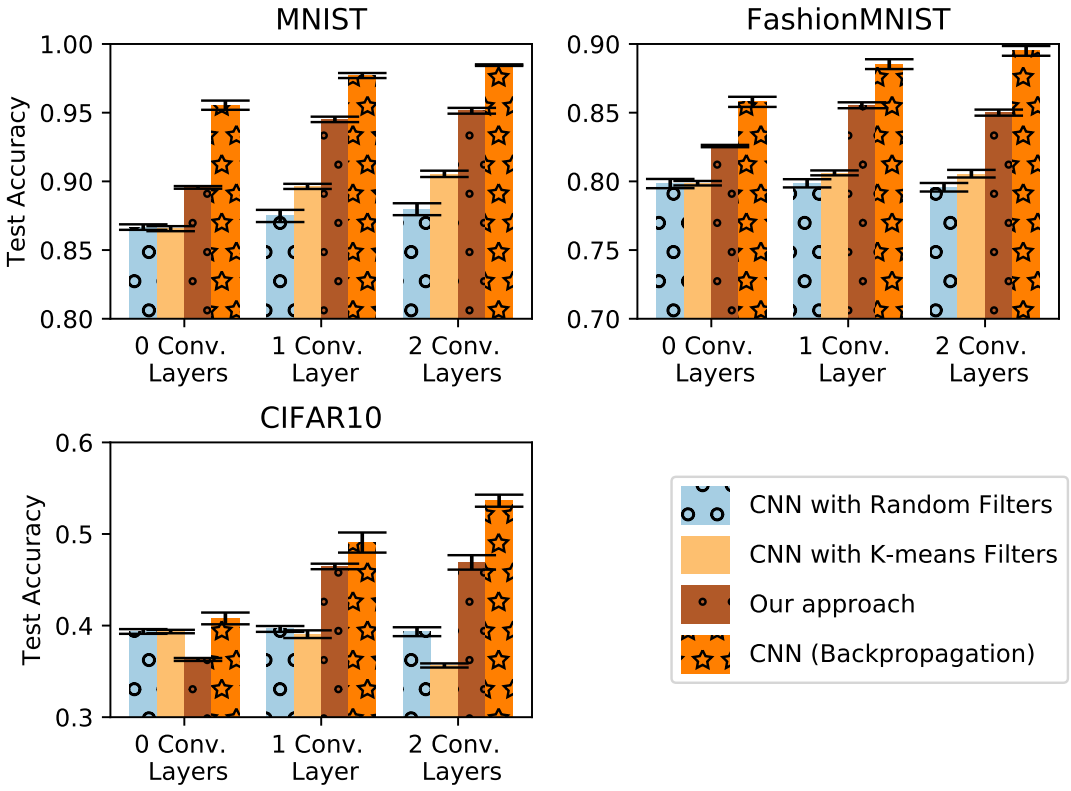

Fig. 2. Comparison of the proposed method with existing training algorithms for CNNs. Each experiment is carried out ten times and we report the mean and standard deviation of the test prediction accuracy for different architecture configurations on several standard image recognition data sets. Our method outperforms existing training algorithms which are not based on gradient-descent for CNNs for almost all architectures and data sets. However, filters learned by backpropagation in CNNs still outperform the analytically determined filters of our approach.

Our experiments show that the filters trained with standard CNNs using a gradient-descent based training algorithm still outperforms our proposed method. We would like to point out, however, that this training method also requires a significant longer training time. In our proof-of-concept setting, we use highly optimized code provided from the keras project[2] with tensorflow backend[3] for the gradient-descent based training and rudimentary optimized code for the non-gradient-descent techniques. Therefore, the training speed comparison is unfair in favor of the gradient-descent based training algorithm. Even for this setting, our approach trains up to 25 times faster than the standard CNN training.

V. Conclusion

In this paper, we proposed a new training scheme for CNNs. Based on the training algorithms from Huang et al. [8], [9] and Qu et al. [19], we showed how the parameters of the filters in convolutional layers of the network can be determined analytically using Moore-Penrose generalized inverses. For that, the same method as for determining the weights in the output layer of a common ELM is used.

We validated our method with several standard image recognition data sets. Our approach achieved significantly higher test prediction accuracy than CNNs with randomly initialized or unsupervisedly learned filters. Moreover, the training is up to 25 times faster with our method than training standard CNNs using the backpropagation algorithm.

An interesting approach for later research could be to combine our algorithm with gradient based methods. We think it would be interesting to investigate whether pre-training with our method followed by gradient-based fine-tuning stabilizes and accelerates the backpropagation training process. Additionally, with the data set size, the memory requirements for our method increase. This can be solved by using special algorithms designed for big data applications. For example, we see methods of randomized linear algebra as promising for reducing memory requirements and computational complexity.

[2]https://keras.io

[3]https://www.tensorflow.org/

References

[1] W. Cao, X. Wang, Z. Ming, and J. Gao. A review on neural networks with random weights. *Neurocomputing*, 275:278–287, 2018.

[2] M. Carreira-Perpinan and W. Wang. Distributed optimization of deeply nested systems. In *Artificial Intelligence and Statistics*, pages 10–19, 2014.

[3] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma. Pcanet: A simple deep learning baseline for image classification? *IEEE transactions on image processing*, 24(12):5017–5032, 2015.

[4] S. Ding, L. Guo, and Y. Hou. Extreme learning machine with kernel model based on deep learning. *Neural Computing and Applications*, 28(8):1975–1984, 2017.

[5] M. M. dos Santos, A. G. da Silva Filho, and W. P. dos Santos. Deep convolutional extreme learning machines: filters combination and error model validation. *Neurocomputing*, 329:359–369, 2019.

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

[7] G.-B. Huang, Z. Bai, L. L. C. Kasun, and C. M. Vong. Local receptive fields based extreme learning machine. *IEEE Computational intelligence magazine*, 10(2):18–29, 2015.

[8] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE international joint conference on neural networks (IEEE Cat. No. 04CH37541)*, volume 2, pages 985–990. IEEE, 2004.

[9] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 498–515. Springer, 2015.

[12] G. Li and P. Niu. An enhanced extreme learning machine based on ridge regression for regression. *Neural Computing and Applications*, 22(3-4):803–810, 2013.

[13] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):1–10, 2016.

[14] M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[15] M. D. McDonnell and T. Vladusich. Enhanced image classification with a fast-learning shallow convolutional neural network. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2015.

[16] S. Pang and X. Yang. Deep convolutional extreme learning machine and its application in handwritten digit classification. *Computational intelligence and neuroscience*, 2016, 2016.

[17] Y.-H. Pao, G.-H. Park, and D. J. Sobajic. Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing*, 6(2):163–180, 1994.

[18] Y.-H. Pao and Y. Takefuji. Functional-link net computing: theory, system architecture, and functionalities. *Computer*, 25(5):76–79, 1992.

[19] B.-Y. Qu, B. Lang, J. J. Liang, A. K. Qin, and O. D. Crisalle. Two-hidden-layer extreme learning machine for regression and classification. *Neurocomputing*, 175:826–834, 2016.

[20] W. F. Schmidt, M. A. Kraaijveld, R. P. Duin, et al. Feed forward neural networks with random weights. In *International Conference on Pattern Recognition*, pages 1–1. IEEE COMPUTER SOCIETY PRESS, 1992.

[21] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein. Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning*, pages 2722–2731, 2016.

[22] D. Xie, J. Xiong, and S. Pu. All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6176–6185, 2017.

[23] M. Yousefi-Azar and M. D. McDonnell. Semi-supervised convolutional extreme learning machine. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1968–1974. IEEE, 2017.