Compressing Convolutional Neural Networks by L0 Regularization

András Formanek
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
formandras+research@gmail.com

Dániel Hadházi
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
hadhazi@mit.bme.hu

Abstract— **Convolutional Neural Networks have recently taken over the field of image processing, because they can handle complex non algorithmic problems with state-of-the-art results, based on precision and inference times. However, there are many environments (e.g. cell phones, IoT, embedded systems, etc.) and use-cases (e.g. pedestrian detection in autonomous driving assistant systems), where the hard real-time requirements can only be satisfied by efficient computational resource utilization. The general trend is training larger and more complex networks in order to achieve better accuracies and forcing these networks to be redundant (in order to increase their generalization ability). However, this produces networks that cannot be used in such scenarios. Pruning methods try to solve this problem by reducing the size of the trained neural networks. These methods eliminate the redundant computations after the training, which usually cause high drop in the accuracy. In this paper, we propose new regularization techniques, which induce the sparsity of the parameters during the training and in this way, the network can be efficiently pruned. From this viewpoint, we analyse and compare the effect of minimizing different norms of the weights (L1, L0) one by one and for groups of them (for kernels and channels). L1 regularization can be optimized by Gradient Descent, but this is not true for L0. The paper proposes a combination of Proximal Gradient Descent optimization and RMSProp method to solve the resulting optimization problem. Our results demonstrate that the proposed L0 minimization-based regularization methods outperform the L1 based ones, both in terms of sparsity of the resulting weight-matrices and the accuracy of the pruned network. Additionally, we demonstrate that the accuracy of deep neural networks can also be increased using the proposed sparsifying regularizations.**

Keywords— compressed sensing, pruning, regularization, L0 minimization, sparsifying regularization

I. Introduction

Although the first Neural Networks (NNs) were implemented in the middle of the previous century [1], the lack of large training data sets and computational resources delayed their big breakthrough in image processing problems for many decades. This period ended in 2012, when a Convolutional Neural Network (CNN), the AlexNet [2] achieved a top-5 error of 15.3% in the ImageNet Challenge beating all other previously proposed methods. Since the success of AlexNet, the area of utilization of CNN has been growing and now, the CNN became a native tool for solving most of image processing related problems.

It has been theoretically and empirically demonstrated [3] that more complex CNNs can be trained to discover and mimic more complex mappings between input and output data. However, in cases when the computational burden is low and real time processing is also a requirement, this approach cannot be directly used. The reason of this is that if we follow this trend, we will not have enough computational capacity to run the trained networks in test times, and the train and storage of such large networks are also problematic.

The knowledge representation of neural networks and especially CNNs is very redundant, which implies that the learnt mappings in given layers of a CNN are highly correlated [4]-[6]. This nature of CNNs can be utilized in parameter number reduction and there are many proposed methods, like pruning, quantization, distillation, which compress the knowledge representation of the networks into less memory- and inference time consuming ones. Most of these methods are applied after training, followed by fine-tuning. In this case, the training of the networks is not conditioned to avoid this redundancy. Additionally, there are widely applied regularization techniques (e.g. Dropout [7]), which directly increase the redundancy of the constructed CNNs. However, other regularization methods can be used as well in order to increase the generalization ability of the CNNs.

In the field of Neural Networks, L2 (weight decay) and L1 (Lasso) are widely used direct regularization methods. Both of them can be successfully optimized by the gradient descent based optimization, like SGD or Adam. L1 is usually utilized in order to generate sparse networks (e.g. [8],[9]). It is demonstrated [10], [11], that from the convex regularization penalty terms, L1 minimization is effective in the sense of sparsification. However, contrary to signal processing problems, the cost function of deep neural networks is always highly non-convex. Therefore, using convex regularizer terms does not imply easier optimization problems, while based on results of Compressed Sensing [12],[13], their effectiveness can be outperformed by non-convex regularizers (e.g. Lp, L0 norm based ones) in the sense of sparsity. Theoretically, minimization of the L0 norm regularized cost function produces the sparsest networks. However, the widely used gradient descent based optimization methods (e.g. SGD, RMSprop, Adam, Adadelta) cannot handle the L0 regularization sensibly, because its gradient is either not defined or 0 in every points of the domain of the weights of the network. In order to overcome this problem, we propose a Proximal Gradient Descent Method [14] (referred hereafter by PGM) based

978-1-7281-3572-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCAIRO47923.2019.00032

adaptation of the RMSprop [15] algorithm for such optimization problems.

In this paper, we introduce a regularization method that can be applied during training time, which cause that the compression of knowledge representation is rewarded during the parameter tuning, instead of forcing it to an already trained model. By this way, we can achieve high compression rate of the network, with also a slight increasing in the accuracy. In our concrete examination (see Sec. III) the accuracy slightly increased when the 80% of the parameters of the network was already eliminated. Our PGM based L0 regularization method can be efficiently implemented and gives better results than L1 regularization.

The rest of the paper is organized as follows. In Section II different kinds of regularizations are compared based on the aspect of pruning, and the adaptation of the Proximal Gradinet Method to the optimization of the cost function is also proposed. Section III compares the examined regularization techniques and optimization algorithm based on quantitative tests. The discussions and the conclusions are presented in Section IV.

II. Materials And Methods

A. Related Works

Pruning methods are designed to reduce the number of parameters of neural networks after training. These methods require a pre-trained model, what they compress iteratively in two steps: 1) drop certain parameters of the network, 2) fine tune the weights of the pruned network to regain accuracy. Two main groups of strategies of choosing the parameters to be pruned can be distinguished: the elements of the first consider only the weights of the network, while the latter ones chose the weights to be eliminated by their effect on the activations. The former methods are usually referred by data independent, while the later ones are called data dependent pruning. According to the selection of the weights, they can be ranked separately (weight pruning), or in groups of weights: coherent weights of kernel matrices (2D filters) or all the weights constituting each channel (3D filters).

Le et al. [16], and Hassibi et al. [17] use second order Taylor series to estimate the effects of pruning in the changing of the activations. Luo et al. [18] and Zhuang et al. [19] propose a greedy algorithm to choose the most important filters of each layer in a CNN. Li et al. [20] utilizes L1 norm to calculate the importance of channels. Wen et al. [21] and He et al. [22] use L2 norms to calculate importance of certain groups of weights - filters, kernels, skip-connections - and L1 norm based optimization to derive sparsity. Lebedev et al. [23] also uses L1,2 regularization, considering the idea of regularized training from scratch, which is followed by pruning and fine tuning. Zhou et al. [24] also minimize L2,1 norm during training (by utilization of proximal gradient method) to prepare the trained model for pruning. In contradiction to these approaches, our proposed method directly minimizes the L0 norm of the weights, which cause higher accuracies with the same number of preserved parameters. The main contribution of this paper is utilization of the direct L0 minimization of the (group of) weights and the adaptation of the Proximal Gradient Descent Method to solve the resulting optimization problem with adaptive learning rate.

Crowley et al. [25] have shown that the L1 norm based pruning is not efficient in sparsifying the weight matrices. The paper demonstrated that training a model from scratch (with the structure that is the result of pruning a larger one), results in a more accurate network, than the pruned one. Our proposed method not only keeps the accuracy at the baseline level but is also able to increase it during the elimination of redundancy in the network.

Please note, that there are other approaches in the state-of-art to reduce the computational resources of neural networks. The two main areas besides pruning are the quantization and low rank factorization of the weight tensors. Courbariaux et al. [26] and Rastegari et al. [27] use binary activation and weight values. Gong et al. [28] use vector quantization on weights of fully connected layers, while Zhao et al. [29] compares using 8-bit integer with 32-bit floating point activations. Sindhwani et al. [30] and Denton et al. [31] propose to learn the parameters of low rank factors of filter matrices.

B. Comparison of regularization methods

Our main goal is constructing a cost function and the corresponding optimization method, which automatically prune the network during the learning. In order to achieve this, the utilization of regularization methods is necessary. In this section we compare different type of regularization methods based on the viewpoint of constructing a prunable, minimally redundant network. For the theoretical comparison, we examine the posterior of the examined regularizations based on its proximal mapping [32]

In the case of supervised learning, the cost function (which is minimized during the training) can be decomposed into sum of the fidelity (or loss) and the regularization (or penalty) terms:

$$C(W) = \sum_i \ell\left(f\left(x_i, W\right), d_i\right) + \rho \cdot \mathcal{H}(W) \tag{1}$$

where C is the cost function, ℓ is the loss function (usually defined by the negative log-likelihood of the network output), $f(x_i, W)$ is the output of the network for x_i input and W weights, d_i is the target (or label) belonging to x_i. ρ is a hyperparameter, which defines the relative importance of the penalty term and $\mathcal{H}(\cdot)$ is the regularization function (usually defined by the logarithm of the density function of the prior distribution of the weights).

Training is basically searching the minimum of C in regard to W:

$$\arg\min_W \left\{C(W)\right\} \tag{2}$$

However, the minima of these functions (realized by CNNs and NNs in general) cannot be computed analytically, therefore we use iterative optimization methods such as Gradient Descent, or other gradient based methods to find a local minimum.

In order to examine and compare the effect of different regularizers, we introduce the Proximal mapping operator:

$$prox_{\mathcal{H},\rho}(W') = \arg\min_{W} \left\{ \frac{1}{\mu} \| W - W' \|_2^2 + \rho \cdot \mathcal{H}(W) \right\} \quad (3)$$

This operator is utilized in Proximal Gradient Descent Methods (PGM) [32] to minimize the regularized cost function alternately with gradient descent update of the variables based on the loss function. The adaptation of the PGM to our optimization problem is discussed later in details (in Sec II.C).

1) Thikhonov regularization:

The penalty function of Tikhonov regularization is the sum of squares of the L2 norm of the weight vectors or the Frobenius norm of the weight kernels in the case of CNN:

$$\mathcal{H}(W) = \sum_{l,i,j} \sum_{u,v} W_{i,j}^{(l)}(u,v)^2 \quad (4)$$

Where $W_{i,j}^{(l)}(u,v)$ is the (u,v)-th element of the i-th kernel matrix of the j-th channel of the l-th layer. This regularization is referred by L2 regularization in NN terminology. The effect of this regularization, based on its proximity operator is:

$$prox_{\mathcal{H},\rho}\left(W_{i,j}^{\prime(l)}(u,v)\right) = W_{i,j}^{\prime(l)}(u,v) \cdot \frac{2}{2 + \mu \cdot \rho} \quad (5)$$

As it can be noticed, from the posterior, and from Fig. 1., this regularization modulates the weights uniformly, therefore it does not trend to introduce sparsity. As it is demonstrated [33], the effect of additive stochastic noise on the activation maps can be modelled by Thikhonov regularization.

2) Dropout:

Dropout is an implicit regularization technique, which was proposed in [7]. During the training, the activations of the layers are gated by multiplying with a scalar, which is sampled from a Bernoulli distribution:

$$\mathbf{y}_i^{(l)} = \sigma\left(\sum_k \mathbf{W}_{k,i}^{(l)} * \mathbf{y}_k^{(l-1)} + b_i^{(l)} \right) \cdot g_i^{(l)} \quad g_i^{(l)} \sim Bern(p) \quad (8)$$

Here $\mathbf{y}_i^{(l)}$ is the i-th output channel of the l-th layer, $\sigma(\cdot)$ is the utilized non-linearity and $g_i^{(l)}$ is the stochastic gate of the examined channel, $Bern(p)$ is the Bernoulli distribution. The values of the gating variables are resampled in the beginning of every epoch. The dropout trends the network to become robust to gating the channels out, therefore the features of the channels become redundant and the whole regularization decreases the sparsity of the kernels in order to become more robust by utilizing this redundancy.

3) Batch Normalization:

Batch Normalizations [34] is originally proposed to increase the learning speed of deep neural networks by reducing the internal covariance shift of the layers. Formally, this means the following layer:

$$y_i = BN_{\gamma,\alpha}(x_i) = \gamma \frac{x_i - \mu_i^{\beta}}{\sqrt{\left(\sigma_i^{\beta}\right)^2 + \varepsilon}} + \alpha \quad (7)$$

where x_i is the input of the layer (e.g. activation of neuron, or an output channel of a convolution layer), μ_i^{β} is its mean in the current minibatch (denoted by β), σ_i^{β} is its standard deviation in β, ε is a small number introduced in order to avoid division by zero. γ and α are learnable parameters of the layer. This layer smooths the surface of the cost function [35], which increases the convergence of the gradient descent methods. Additionally, it introduces regularization as side effect, because the value of μ_i^{β} and σ_i^{β} is the function of the training samples of the current minibatch. From this viewpoint, it can be considered as the mixture of the stochastic L2 (additive noise) and the Dropout regularization (multiplicative noise), which induces more robust and more redundant networks. Therefore, this regularization also does not motivate the network to learn sparse weights, however its utilization in deep networks is unavoidable if we want to achieve state-of-the-art precision in practice. This is the reason, why our networks architecture in the Sec. III also contains such layer.

4) L1 norm minimization (Lasso [8]):

L1 is another widely used regularization in this field. The penalty function in this case, similarly to L2, is the L1 norm - the sum of the absolute values - of the weights:

$$\mathcal{H}(W) = \sum_{l,i,j} \sum_{u,v} \left| W_{i,j}^{(l)}(u,v) \right| \quad (8)$$

This regularization is referred by L1, Lasso, shrinkage regularization and explicitly utilized in order to sparsify the networks. This is the consequence of the fact that contrary to L2, the low values are penalized more, and in this way, the optimization trends to produce zero weights. This can also be conducted from the corresponding proximity operator:

$$prox_{\mathcal{H},\rho}\left(W_{i,j}^{\prime(l)}(u,v)\right) = \max\left\{ 0, | W_{i,j}^{\prime(l)}(u,v) | - \frac{\rho}{2\mu} \right\} \cdot sign\left(W_{i,j}^{\prime(l)}(u,v)\right) \quad (9)$$

This operator is the so-called shrinkage or soft-threshold operator, which reduce the norm of the weights or the kernels additively (see also Fig. 1.). Therefore, it zeros out weights, or group of weights, whose L1 norm is less than ρ/2μ. Please note that ideal sparsity regularization would be the L0 norm based one. L1 approximates L0 generally with the lowest error from the convex functions [10], but cannot produce as sparse weights as the L0 based can [12]. Due to the fact, that in the case of machine learning, the whole optimization problem is also non-convex and usually non-differentiable (in contradiction to classical image and signal processing problems [11]), the utilization of convex penalty term is not a big advantage in deep learning.

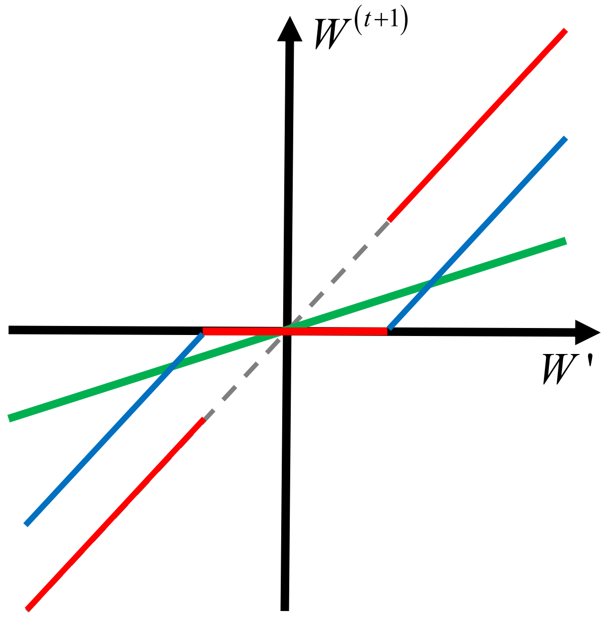

Figure 1: Posteriror of the examined direct regularization techniques (based on the corresponding proximal mapping). The curves show the output of the regularizations (green – Tikhonov, blue – L1, red – L0) as function of their input weight.

5) L0 norm regularization:

The L0 norm of a weight is the number of its non-zero elements:

$$\mathcal{H}_0(W)=\sum_{l,i,j}\sum_{u,v}1-\delta\left(W_{i,j}^{(l)}(u,v)\right) \tag{10}$$

where δ(·) is the Kronecker delta function (sometimes referred as indicator function). Therefore, this penalty function penalizes all the non-zero weights of the network uniformly. The proximity operator of this regularization is:

$$prox_{\mathcal{H},\rho}\left(W_{i,j}^{'(l)}(u,v)\right)=\begin{cases}0 & \left\|W_{i,j}^{'(l)}(u,v)\right|<\rho\mu \\ W_{i,j}^{'(l)}(u,v) & \left\|W_{i,j}^{'(l)}(u,v)\right|\geq\rho\mu\end{cases} \tag{11}$$

As it can be concluded from the operator and Fig. 1., the L0 regularization zeroes out the weights that have smaller amplitudes than $\rho\mu$ and does not change the value of the others. Based on the results of Compressed Sensing [12], L0 reduces the number of non-zero weights more effectively, compared to L1, with reaching the same accuracy. This is the reason, why we propose the utilization of this regularization. Pleas also note that this regularization does not change the value of bigger weights, in contradiction to the L1 based one, which reduce the amplitudes of the weights uniformly. Structured regularization:

The above regularizations are all defined to treat the weights separately. Our final goal with regularization is to prepare networks, that have reduced memory and computational resource requirements. Effectively utilizing weight pruning based on these aspects is generally not an easy task. The reason is the fact that the successfully pruned values are usually scattered amongst the weight tensors. Thus, the sparse kernels also have to be stored and the number of necessary convolution operations in inference still not changes. Structured pruning is much more desirable, because it can reduce the number of non-zero kernels or output channels of a convolutional filter.

LF norm based pruning considers the Frobenius norm of a kernel/channel as the importance of it. The penalty terms are:

$$\mathcal{H}_{\ker,p}(W)=\sum_{l,i,j}\varphi_p\left(\sqrt{\sum_{u,v}\left(W_{i,j}^{(l)}(u,v)^2\right)}\right) \tag{12}$$

$$\mathcal{H}_{chan,p}(W)=\sum_{l,j}\varphi_p\left(\sqrt{\sum_{i,u,v}\left(W_{i,j}^{(l)}(u,v)^2\right)}\right) \tag{13}$$

here $\mathcal{H}_{\ker,p}(W)$ corresponds to the kernel-wise regularization, while $\mathcal{H}_{chan,p}(W)$ is the penalty term of the channel-wise regularization, while the sub-script p corresponds to the utilized norm. In this paper, we examine the extension of the L1 and L0 regularization:

$$\varphi_p(a)=\begin{cases}|a| & |p=1 \\ 1-\delta(a) & |p=0\end{cases} \tag{14}$$

The corresponding proximity operator is similar to (9) and(11). In the case of the channel-wise regularization, the inference speed in test time can be easily increased without modification of the computational framework (because in this case, the pruned network is a classical CNN). However, it reduces the size of the hypothesis space much more compared to the kernel-wise regularization (with the same number of weights, or computational resource requirement). Our empirical results demonstrate the same. Therefore, it should be deliberated, which grouping is preferable in the concrete use-case.

C. Optimization of the L0 regularized cost function

Based on the comparison of the different, commonly applied regularization techniques, we chose the utilization of the L0 based ones (the weight, the kernel and the filter –wise L0 norms) to construct sparse neural networks. The minimization of such regularizer with Gradient Descent methods is extremely hard, because the derivative of the penalty term is 0 everywhere:

$$\partial\mathcal{H}_0(W)=0 \tag{15}$$

where ∂ denotes the sub-gradient operator. Therefore, the gradients and the sub-gradient of the error surface does not provide any useful information about how to minimize it. The common feature of the already published techniques for similar problems is that the penalty term is relaxed, therefore the classical gradient descent can be used for solving the optimization problem. We propose different solution to solve these problems, the utilization of the Proximal Gradient Method (PGM) [32]. PGM is designed to minimize cost functions, which can be decomposed into two parts. To a term (like the *loss* term in(1)) that can be minimized effectively by gradient descent method. And to a second term (like the *penalty* term in (1)), which has easily computable proximity operator. The method is convergent for K-L functions [36], which set contains the cost functions of typical networks. An iteration of a PGM is:

$$\begin{aligned} W' &= W^{(t)} - \mu \cdot \nabla_W \sum_i \ell\left(f\left(x_i, W\right), d_i\right) \\ W^{(t+1)} &= prox_{\mathcal{H},\rho}\left(W'\right) \end{aligned} \tag{16}$$

where ∇ is the gradient operator, μ is the learning-rate. The first equation defines a traditional gradient descent step based on the *loss* term, while the posterior of the regularization is calculated based on the proximity operator (which we described for different kind of regularizations above).

Similar to optimization of regular cost functions, the estimation of the gradient of the loss term has too high computational burden, which can be reduced by stochastic optimization. In this case, the first step of the PGM iterations are changed to the following:

$$W' = W' - \mu \cdot \nabla_W \sum_{i \in \beta_j} \ell\left(f\left(x_i, W\right), d_i\right) \tag{17}$$

where β_j is the set of the indexes of the training samples in the j-th mini batch. Adaptive gradient methods can also be used to reduce the value of the loss term like Adagrad [37], RMSProp [15], Adadelta [38]. In this paper, we propose the adaptation of the RMSProp to the optimization problem. In this case (the method hereafter referred by Proximal RMSProp), the first row of (16) is replaced by:

$$\begin{aligned} q^{(j)} &= \upsilon \cdot q^{(j-1)} + (1-\upsilon) \cdot \nabla_W \sum_{i \in \beta_j} \ell\left(f\left(x_i, W'^{(j-1)}\right), d_i\right)^2 \\ W'^{(j)} &= W'^{(j-1)} - \frac{\mu \cdot \nabla_W \sum_{i \in \beta_j} \ell\left(f\left(x_i, W'^{(j-1)}\right), d_i\right)}{\sqrt{q^{(j)}} + \in} \end{aligned} \tag{18}$$

and $W' = W'^{(M)}$ and $W'^{(0)} = W^{(t)}$ in the $(t+1)$-th iteration. Here M is the number of the batches. The division, the squaring and the square root is calculated element wisely. The normalizer, mean square term ($q^{(j)}$) is initialized only in the first iteration of t. Please note, that by this definition, the penalty term is not considered in the definition of q. This is preferred in the case of L0 based regularizations, because the proximal operator either does not change a weight or it zeros it out (see(11)). If we modify the value of q only at the end of an iteration of t, than the adaptive learning rate of the zero outed weights will monotonically increase, which induce numerical stability problems for larger value of μ. Therefore, this can slow down the whole training and reduce the sparsity of the weights. We also noticed that the proposed optimization method does not make impossible for the pruned out weights to become non-zero during the training in later iterations. Thus, the architecture of the network is continuously changing during the training.

The weight of the regularization is chosen uniformly when regularizing weights or kernels in default. We found two main drawback of this approach. First, it is very hard to estimate the size of the resulting network. In the case of L1 regularization, zero weight values never appear, only weights with small amplitudes. This induces the necessity of finding a threshold for pruning the network after the end of the training. In the case of using L0, the pruning is trivial, because the nulled weights can be dropped out without any problem. The second problem with uniform regularization parameter is rather practical, in case of regularizing whole channels with L0 it drops significantly more weights from the first convolutional layer compared from the later ones. Both problems can be eliminated if instead of applying uniform weights, we define a compress ratio and chose the regularization parameter of the layers separately to satisfy this. In case of n% compression with L0, this means that the $\rho\mu$ for a given layers is always the n-th percentile of the Frobeinus norms of the weight matrix of the channels.

First order momentum can also be used in calculating PGM optimization, but in that case, the effect of the extrapolation step must be checked in every iteration, which induces significant overhead into the optimization (see [39] for details). Based on our recent experiments, this overhead is higher, than the acceleration, which can be realized by extrapolation along the first momentum, therefore we do not suggest the utilization of this approach.

III. Results

In this section, we examine and compare the effect of L0 and L1 weight, kernel and channel-wise minimization based regularizations. For this test, we used the CIFAR-10 [40] dataset, which is a widely used database for Image Processing benchmark problems. This dataset contains 50,000 training and 10,000 test samples from 10 classes. The size of the images is 32×32×3. We trained a VGG [41] variant, which is originally designed for the ImageNet dataset, but also widely used for other benchmarks. The feed-forward network contains 8 Conv2D blocks (which contains a Conv2D, a Batch-Norm and a ReLU layer in this order) and 4 MaxPooling layers.

The architecture of the network is described in Table I. M denotes MaxPooling (2×2), while *C(n)* denotes Conv2D blocks with *n* number of output channels, *FC(m)* denotes fully connected layers with *m* neurons. The network contains 9,242,730 trainable weights. During our measurements, the regularization was only applied on the weights of the Conv layers.

TABLE I. ARCHITECTURE OF THE EXAMINED VGG VARIANT:

C*(64), M, C(128), M, C(256), C(256), M, C(512), C(512), M, C(512), C(512), M, FC(32), FC(10)*

During training, the size of one batch was 256, early stopping was used with patient value of 8. For the optimization of L1 regularized cost functions, we used

RMSProp method with $\upsilon = 0.9$ and $\mu = 1E-3$. For optimizing the L0 regularized cost functions, Proximal RMSProp was utilized with the same values of hyper-parameters ($\upsilon = 0.9$ and $\mu = 1E-3$). The loss function was *categorical cross entropy* in all tests.

We compared three above described regularization method L1, L0 with uniform regularization weight (referred by *L0 with uniform threshold)* and L0 with compression rate (referred by *L0 with compression rate*). As a baseline, we trained the network without any regularization (except the BatchNorm layers induced one) 45 times from random Xavier [42] initialization. The reached accuracies are summarized in Table II.

TABLE II. ACCURACY OF BASELINE MODELS:

	Train acc.	Valid acc.	Test acc.
Mean	93.4198%	79.2191%	78.1929%
Variance	0.2686%	0. 0722%	0.0728%

A. *Weight-wise regularization*

In case of weight-wise regularization we found that contrary to L1, both of the L0 methods can increase the accuracy of the baseline network at lower rates of compression (40%-80%). This confirms that L0 utilizes the available resources more efficiently. We found the tuning of the regularization parameter of L1 extremely hard, meaning that there is a very narrow gap between ρ -s induces dropping more than 80% of weights or less than 5%.

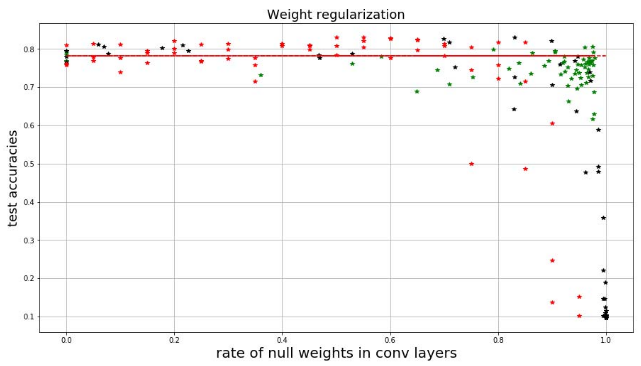

Figure 2: Results of regularizing weight-wise (not in groups). Green: L1 regularization, black: L0 with uniform threshold, red: L0 with compressing rate.

Both *L0 with compressing rate* and *L0 with uniform threshold* can increase the test accuracy if less than 80% of the weights are pruned during training. While the L1 gives the best results if higher percent (>90%) of weights are dropped out. The accuracy of the pruned networks in the function of the compression rate is shown in Fig.2.

B. *Kernel-wise regularization*

During the kernels-wise L1 regularization we detected the same difficulties as during weight-wise: the number of dropped kernels is either less than 20% or higher than 80%, depending on the value of the regularization parameter. This effect is not present in the case of using either L0 variant.

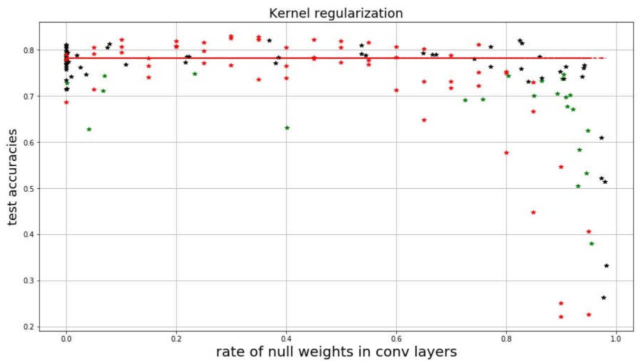

Figure 3 Results of kernel-wise regularization. Green: L1 regularization, black: L0 with uniform threshold, red: L0 with compressing rate.

As we can see on Fig3, like weight-wise, kernel-wise L0 regularization can also increase the test accuracy of the network in lower compression rates (<0.8), while L1 regularization cannot. L1 consistently reduces accuracy with 2-3%, while both L0 based methods can increase it with 2-3% in wide range of ρ values. Note that in this case, *L0 with uniform threshold* also outperforms L1 in the reduction of the number of parameters. The best model with more than 90% of compression rate preserves only 5.75% percent of the weights and reaches 76.72% percent test accuracy, while the best model of this kind of networks constructed by L1 regularization leaves 9.51 % of the weights not nulled and reaches only 74.6% test accuracy.

C. *Channel-wise regularization*

As we mentioned above, *L0 regularization with uniform threshold* is not effective in this use-case. Thus, only the using of L1 or *L0 with compression rate* is considerable. In the case of L1, the accuracy suddenly decreases at 80% compression rate, which is not true for L0. In lower compression rates, the L0 regularizer also induces higher test accuracies.

As can be seen from Fig.4, channel pruning can only slightly increase accuracy, but L0 with compression rate slightly outperforms L1 in this aspect too.

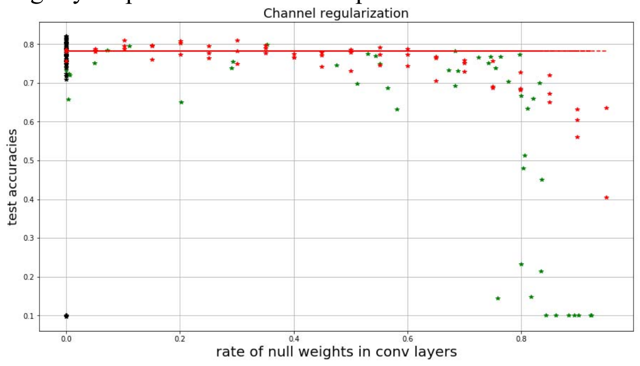

Figure 4 Results of channel-wise regularization. Green: L1 regularization, black: L0 with uniform threshold, red: L0 with compressing rate.

IV. Discussion and conclusion

In this paper we have shown that L0 norm based regularization techniques can be utilized to training neural networks in order to compress them. We found that this compression can be realized with slight increment in test accuracy. This is in contradiction to the results of [25], which examined L1 norm based methods from this aspect. We proposed an adaptation of the Proximal Gradient Method (PGM) to solve the minimization problem of the L0 regularized cost functions. We also proposed a combination of the PGM and the RMSProp (referred by Proximal RMSProp), which can be effectively used in the optimization of the proposed L0 regularized cost functions of the neural networks.

We have also empirically shown and theoretically discussed that L0 regularization has numerous benefits compared to the widely used L1 norm based relaxation that is often used to increase the sparsity of the parameters. The proposed regularization and the corresponding optimization method can be easily implemented in the most frequently used frameworks (like TensorFlow, Pytorch, Theano). It also does not increase the number of floating point operations, which is required during the optimization per epochs compared to the L1 regularized case.

According to the results that correspond to weight-wise regularization, the L0 based techniques can produce networks with higher accuracies compared to the baseline. We also observed that, counter to our previous assumptions, L1 can zero out more weights with relatively lower drop in accuracy at very large compressing rates (e.g. drop more than 95% of weights). However, this benefit cannot accelerate the inference time, because weight-wise pruning generally does not decrease the number of required operations in the case of CNNs. Thus, in this use-case, increasing the generalization-ability is much more relevant during learning the architecture of the network (in order to increase the test accuracy). Based on this consideration, the L0 regularizations can prove better results (higher test accuracies).

Kernel-wise pruning is not often discussed by the state-of-the-art. The most important reason of this is that current DeepLearning frameworks cannot efficiently utilize this kind of sparsity (however, in contradiction to the weight-wise sparsity, it reduces the number of required convolutions in inference time). As far as this does not change, the most important goal of kernel-wise pruning is learning architecture of the network in order to achieve better test accuracies. The advantage of applying kernel-wise regularization during train time is that it motivates the network to sparsify the connections between channels of adjacent layers, which can facilitate higher specialization of the channels. This is just the opposite of what DropConnect [43] is doing in order to achieve the same goal. We have demonstrated that at around 50% compression rate, the accuracy of the network is higher than that of the initial model. In the case of kernel-wise learning, L0 is also more effective in parameter number reduction.

Channel-wise pruning is practically the most important pruning approach based on the reduction of inference time- and memory requirements. The reason of this is that every channel without any non-zeros weight can be omitted from the computational graphs without the modification of the neural network realized mapping. In addition, the architecture of the pruned CNN is also a conventional one (with less parameters), which can be efficiently handled by nowadays frameworks. Our empirical results shown that in the case of channel-wise regularization, L0 is also more effective than L1. This means that higher accuracies are reached in the whole range of the compression rate by L0 with compressing rate regularization.

In the future, we plan to examine how the Adam can be more efficiently combined with the Proximal Gradient Methods (e.g. take a step in the direction of the first momentum and the necessary check of the extrapolation after every n-th mini batch). It is also an interesting question how the L0 regularization can compress networks with dropout layers, how can it be adapted to other special kind of layers (e.g. residual block, attention mechanism, etc.) and how does these type of layers modify the performance of the regularization.

Acknowledgment

This research has been supported by the EFOP 3.6.1.-16-2016-00014 program (Research, development, and educational integration of disruptive technologies in the field of e-mobility), co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013), the BME-Artificial Intelligence FIKP grant of EMMI (BME FIKP-MI/SC) and the Nvidia Gpu Grant program.

References

[1] Ivakhnenko, A.G. „*Cybernetics and forecasting techniques,*" New York, American Elsevier (1967).

[2] A. Krizhevsky, I. Sutskever, G. E. Hinton. „*ImageNet Classification with Deep Convolutional Neural Networks,*" NIPS (2012).

[3] G. Montufar, R. Pascanu, K. Cho, and Y. Bengio. „*On the number of linear regions of deep neural networks,*" NIPS (2014).

[4] M. Denil, B. Shakibi, L. Dinh, and N. de Freitas. „*Predicting parameters in deep learning,*" NIPS (2013).

[5] A. Coates, A. Y. Ng, and H. Lee. „ *An analysis of single-layer networks in unsupervised feature learning,*" Artificial Intelligence and Statistics (2011).

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. „*Imagenet classification with deep convolutional neural networks,*" NIPS (2012).

[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. „*Dropout: A Simple Way to Prevent Neural Networks from Overfitting,*" Journal of Machine Learning Research, vol.: 15, pp.:1929-1958 (2014).

[8] Tibshirani, Robert. „*Regression Shrinkage and Selection via the lasso,*" Journal of the Royal Statistical Society. Series B (methodological). Wiley. vol.: 58(1), pp.: 267–88, .1996.

[9] H. Z. Alemu, J. Zhao, F. Li, and W. Wu. „*Group L1/2 Regularization for Pruning Hidden Layer Nodes of Feedforward Neural Networks,*" IEEE Access, vol. 7, pp. 9540–9557, 2019.

[10] Ramirez C, Kreinovich V, Argaez M. „*Why ℓ1 is a good approximation to ℓ0: A geometric explanation,*" J Uncertain Syst. (2013).

[11] E. J. Candes and M. B. Wakin. „*An Introduction To Compressive Sampling,*" IEEE Signal Processing Magazine, vol. 25, no. 2, pp. 21–30, Mar. 2008. (2008).

[12] E. J. Candès, M. B. Wakin, and S. P. Boyd. „*Enhancing Sparsity by Reweighted ℓ 1 Minimization,*" Journal of Fourier Analysis and Applications, vol. 14, no. 5–6, pp. 877–905, Oct. 2008. (2008).

[13] H. Zou. „*The Adaptive Lasso and Its Oracle Properties,"* Journal of the American Statistical Association, vol. 101, no. 476, pp. 1418–1429, Dec. 2006. (2006).

[14] Combettes, Patrick L.; Pesquet, Jean-Christophe (2009*). „Proximal Splitting Methods in Signal Processing,"*. arXiv: 0912.3522 (2010).

[15] F. Zou, L. Shen, Z. Jie, W. Zhang, W. Liu. „*A Sufficient Condition for Convergences of Adam and RMSProp,"* In IEEE CVPR 2019, pp. 11127-11135 (2019).

[16] Y. LeCun, J. S. Denker, and S. A. Solla. „*Optimal brain damage,"* NIPS, pp. 598–605 (1990).

[17] Hassibi, B., Stork, D.G. „*Second order derivatives for network pruning: Optimal brain surgeon,"* NIPS (1993).

[18] J.-H. Luo, J. Wu, and W. Lin. Thinet. „*A filter level pruning method for deep neural network compression,"* ICCV, pp.: 5058–5066 (2017).

[19] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo1, Q. W, J. Huang, J. Zhu. *"Discrimination-aware Channel Pruning for Deep Neural Networks,"* arXiv: 1810.11809 (2018).

[20] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. „*Pruning filters for efficient convnets,"* arXiv preprint arXiv:1608.08710, 2016.

[21] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. „*Learning structured sparsity in deep neural networks,"* In Advances In 9 Neural Information Processing Systems, pages 2074–2082 (2016).

[22] Y. He, X. Zhang, and J. Sun. „*Channel pruning for accelerating very deep neural networks,"* ICCV, pp. 1389–1397 (2017).

[23] V. Lebedev, V. Lempitsky. „*Fast convnets using groupwise brain damage,"* In IEEE CVPR 2016, pp. 2554-2564 (2016).

[24] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: „*Towards compact cnns,"* ECCV, pp.: 662–677. Springer International Publishing (2016).

[25] E. J. Crowley, J. Turner, A. Storkey, M. O'Boyle, „*A Closer Look at Structured Pruning for Neural Network Compression*," arXiv: 1810.04622 (2018).

[26] M. Courbariaux, Y. Bengio. Binarynet. „*Training deep neural networks with weights and activations constrained to +1 or -1,"* CoRR (2016).

[27] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. „*Xnor-net: Imagenet classification using binary convolutional neural networks,"* ECCV, pages 525–542 (2016).

[28] Gong, Y., Liu, L., Yang, M., Bourdev, L. „*Compressing deep convolutional networks using vector quantization,"* arXiv:1412.6115 (2014).

[29] D. D. Zhao, F. Li, K. Sharif, G. M. Xia, Y. Wang. „*Space efficient quantization for deep convolutional neural networks,"* Journale Of Combuter Science and Technology vol. 34, no. 2, pp. 305–317 (2019).

[30] V. Sindhwani, T. Sainath, and S. Kumar. „*Structured transforms for small-footprint deep learning,"* NIPS, pp. 3088– 3096 (2015).

[31] E. Denton, W. Zaremba, J. Bruna, Y. LeCun and R. Fergus, „*Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation,"* NIPS (2014).

[32] N. Parikh and S. Boyd, „*Proximal algorithms,"* Found. Trends Optim., vol. 1, no. 3, pp. 123–231 (2013).

[33] C. Bishop. „*Pattern Recognition and Machine Learning,*", Springer-Verlag New York, 2006.

[34] S. Ioffe and C. Szegedy. Batch normalization. „*Accelerating deep network training by reducing internal covariate shift,*" ICML (2015).

[35] S. Santurkar, D. Tsipras, A. Ilyas and A. Madry. „*How Does Batch Normalization Help Optimization,*" NIPS (2018).

[36] D. Noll. „*Convergence of Non-smooth Descent Methods Using the Kurdyka–Łojasiewicz Inequality,"* Journal of Optimization Theory and Applications, vol. 160, no. 2, pp. 553–572, Sep. 2013. (2013).

[37] J. Duchi, E. Hazan, Y. Singer. „*Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,"* JMLR vol. 12. pp. 2121−2159 (2011).

[38] M. D. Zeiler. „*Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,"* arXiv: 1212.5701 (2012).

[39] H. Li and Z. Lin. „*Accelerated Proximal Gradient Methods for Nonconvex Programming,"* NIPS (2015).

[40] A. Krizhevsky. „*Learning Multiple Layers of Features from Tiny Images,"* Tech Report (2009).

[41] K. Simonyan and A. Zisserman. „*Very deep Convolutional Neural Networks for Large-Scale Image Recognition,"* ICLR (2015).

[42] X Glorot, Y. Bengio. „*Understanding the difficulty of training deep feedforward neural networks,"* AISTATS (2010).

[43] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, R. Fergus. „*Regularization of Neural Networks using DropConnect,"* PMLR vol. 28, no. 3, pp.1058-1066 (2013).