Learning the Structure of Deep Architectures Using ℓ_1 Regularization

Praveen Kulkarni[1]
Praveen.Kulkarni@technicolor.com
Joaquin Zepeda[1]
Joaquin.Zepeda@technicolor.com
Frederic Jurie[2]
frederic.jurie@unicaen.fr
Patrick Pérez[1]
Patrick.Perez@technicolor.com
Louis Chevallier[1]
Louis.Chevallier@technicolor.com

[1] 975 avenue des Champs Blancs, CS 17616, 35576 Cesson Sévigné, France
http://www.technicolor.com

[2] University of Caen Basse-Normandie, CNRS UMR 6072, ENSICAEN, France
http://www.unicaen.fr

Abstract

We present a method that formulates the selection of the structure of a deep architecture as a penalized, discriminative learning problem. Up to now, the structure of deep architectures has been fixed by hand, and only the weights are learned using discriminative learning. Our work is a first attempt towards a more formal method of deep structure selection. We consider architectures consisting only of fully-connected layers, and our approach relies on diagonal matrices inserted between subsequent layers. By including an ℓ_1 norm of the diagonal entries of said matrices as a regularization penalty, we force the diagonals to be sparse, accordingly selecting the effective number of rows (respectively, columns) of the corresponding layer's (next layer's) weights matrix. We carry out experiments on a standard dataset and show that our method succeeds in selecting the structure of deep architectures of multiple layers. One variant of our architecture results in a feature vector of size as little as 36, while retaining very high image classification performance.

1 Introduction

Since Krizhevsky *et al.* [10] demonstrated the outstanding results that can be obtained in image classification by using Deep Neural Networks (DNNs), the popularity of DNNs has exploded. DNN methods have indeed resulted in very large increases in performance on a wide range of image classification datasets, including large scale datasets such as ImageNet [10], and, by means of transfer learning, smaller datasets such as PascalVOC 2007/2012 [15], SUN397 [7] and MIT Indoor datasets [3, 7].

DNNs consist of concatenations of standard layers. The first several layers are usually *convolutional layers* consisting each of n_j (with j the layer index) spatially-convolutional kernels that operate on the n_{j-1} dimensional spatial signals output by the previous layer.

Pages 23.1-23.11
DOI: https://dx.doi.org/10.5244/C.29.23

Interspersed between these layers are normalization layers and spatial max-pooling layers that can be seen as non-linear convolutional operators. The normalization layers process a single spatial position and effectively balance kernel output energy across space. The max-pooling layers reduce the spatial support of the signal by max-pooling signals in small spatial neighborhoods. Given the convolutional nature of all these operators, they can process input images of arbitrary dimensions.

The latter layers of DNN architectures are instead fully connected layers that require inputs of fixed size. This constraint on input size propagates down the convolutional layers, effectively fixing the DNN architecture's expected input image size (sizes around 225×225 are common [9, 10]). In this respect, the convolutional layers of DNN architectures can be seen as very large fully connected layers that have been constrained to have a weights matrix that is structured and highly sparse, effectively regularizing the architecture.

One important consideration when designing DNN architectures is the choice of weights matrix size across the various Fully Connected Layers (FCLs). Besides being another important means of regularization, the size of the FCL weight matrices has a strong impact on system complexity. The first fully connected layer, in particular, can account for 90% of the number of coefficients in the DNN [10]. When using pre-learned DNNs as generic image feature extractors [2, 6, 17], rectangular weights matrices further have the potential advantage of producing reduced-size feature vectors [2, 17]. But, up to now, the approach used to select the dimensions of the rectangular matrices across the various layers has been empirical, and researchers have mostly focused on using constant sizes across all fully connected layers.

The main contribution of the present work is hence to show that the sizes of the FCL weight matrices can be selected as a part of the supervised DNN learning procedure. We do this by inserting a diagonal matrix $\mathbf{D}^j$ between layers $(j, j+1)$, accordingly penalizing the DNN learning objective with the sparsity inducing ℓ_1 norm on the diagonal coefficients of each $\mathbf{D}^j$. Our method can be seen as formalizing the tradeoff between the generalization power of the model and its storage/computation requirements, as represented by the FCL weight matrix sizes. We then show experimentally that it is indeed possible to choose optimal FCL weight matrix sizes and that these vary with the layer index. Our experiments further show that the approach results not only in smaller feature vectors, but also in higher image classification performance.

The remainder of this paper is organized as follows: in the next section, we present a review of DNN methods that are related to our work. In Section 3, we present our proposed method, subsequently evaluating it experimentally in Section 4. We then provide some concluding remarks in Section 5.

2 Background

Given the large number of free parameters in DNN architectures, regularization is an important consideration when learning deep architectures, and various works have explicitly addressed it. One very successful approach currently deployed in publicly available DNN learning algorithms [9] is *dropout* [20]. Dropout aims to prevent neighboring units from memorizing training samples. This is accomplished by randomly zeroing a subset of the activation values at the output of each fully connected layer, choosing a different random subset for each training example. The end result is that a different effective training set is used for neighboring weights. A related approach, DropConnect, zeros the FCL weights instead of

the activation coefficients [22]. Other regularization approaches instead consist of using unsupervised learning either as an initializing stage before a second, strongly supervised stage, or in a mixed supervised/unsupervised approach [5].

A related line of work consists of pre-learning an entire architecture on a large, generic image dataset and then adapting the resulting architecture to a new target dataset. The approach of [2], for example, adapts the entire architecture by continuing the learning process at a reduced learning rate using the samples from the new target dataset. In [15], the adaptation is instead carried out by entirely re-learning the last two layers on the new target dataset. A similar idea consists of using the activations of the penultimate layer of a pre-learned architecture as an image feature. When used for image classification, SVM classifiers are then learned on top of these features [17], and the collection of such linear classifiers can be seen as a single fully connected adaptation layer. Linear classifiers have also been used in place of the soft-max classifier when learning either the entire architecture [2], or along with one other fully-connected adaptation layer [12].

Various authors have also considered DNN variants of classical approaches including the ubiquitous spatial pyramid [13], the Fisher kernel [16] and bag-of-words [19]. The approach of [11], for example, consists of treating the n_j-dimensional spatial signals at the output of the j-th layer as a densely extracted local descriptor, and then building an aggregated representation such as a bag-of-words or a Fisher vector on top of these local descriptors.

The work of [8] uses a Spatial Pyramid Pooling (SPP) layer between the last convolutional layer and the first fully-connected layer. This should make it possible to use input images of arbitrary size, as the SPP layer maps the arbitrarily-sized output from the last convolutional layer to a vector of constant size compatible with the subsequent fully connected layer. In practice, however, the approach is implemented using a max-pooling layer with large stride.

The approach presented in [6] consists of using a DNN as a local feature extractor by treating each patch from a dense sampling of image patches as an image. The activation features resulting from each patch are then treated as local descriptors to build an aggregated, global feature vector. A similar approach is presented in [4]: given a large number of region proposals derived from the input image, the method extracts CNN activation features for each region and classifies them into given set of classes using linear classifiers. Approaches such as that of [4, 6] that use DNNs as local feature extractors suffer from a very large computational complexity, and could hence benefit from complexity constrained DNNs such as the ones presented herein.

3 Learning the structure of deep architectures

In this section we present our proposed approach, illustrated in Fig. 1. The architecture we consider consists of a sequence of fully-connected layers, with a diagonal matrix between them. We will constrain the diagonal matrix to have a sparse diagonal, and this will implicitly define the size of the weights matrices of each layer. Rather than using the standard soft-max classification layer as the last layer, we will use a bank of linear SVM classifiers similarly to the approaches in [2, 12].

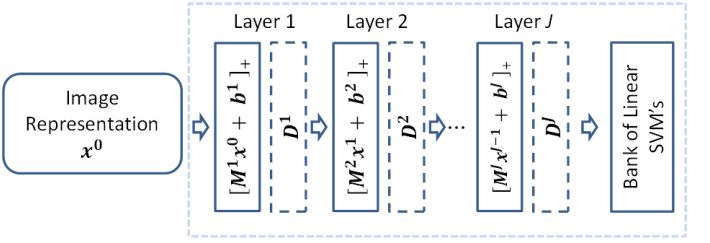

Figure 1: **Proposed deep processing pipeline**. Given an image representation, e.g., the output of the convolutional part of a pre-trained state-of-art DNN, J fully connected layers, each involving a diagonal matrix that controls its effective dimensions, are jointly learned with final linear SVM classifiers. Here, $[\mathbf{z}]_+ = [\max(0, z_i)]_i$ is the commonly used Rectified Linear Unit (ReLU) non-linearity.

3.1 Constraining layer complexity

Formally, we can express the architecture in Fig. 1 as a concatenation of units of the following form:

$$f^j(\mathbf{x}) = \mathbf{D}^j \left[\mathbf{M}^j\mathbf{x} + \mathbf{b}^j\right]_+, \tag{1}$$

where $[\mathbf{z}]_+ = [\max(0, z_i)]_i$ is the commonly used Rectified Linear Unit (ReLU) non-linearity, here applied to a vector $\mathbf{z} = [z_i]_i$. Input vector is n_{j-1}-dimensional and output is n_j-dimensional. This layer is defined by n_j-dim vector $\mathbf{b}^j$, n_j-dim diagonal matrix $\mathbf{D}^j$ and matrix $\mathbf{M}^j$ of size $n_j \times n_{j-1}$. A deep architecture can be derived from (1) using the standard stacking approach. Letting $\circ$ denote the composition operator such that $f \circ g(\mathbf{x}) = f(g(\mathbf{x}))$, this can be denoted as

$$f^J \circ \ldots \circ f^1(\mathbf{x}), \tag{2}$$

where, in this case, the vector $\mathbf{x}$ denotes the representation of the image at the input of the architecture. The image representation can consist of a direct re-ordering of the RGB values in the image [2, 10], or it can be a feature derived from the image [12, 15], which is the approach we follow in the present work.

We are interested in the case when the diagonal entries of $\mathbf{D}^j$ are sparse. When this is the case, the corresponding rows of $\mathbf{M}^j$ and $\mathbf{b}^j$ can be removed, as well as the corresponding columns of $\mathbf{M}^{j+1}$. To see this, let $\mathcal{I}$ denote the support (indices of non-zero positions) of the diagonal entries of $\mathbf{D}^j$. Let $\mathbf{A}_{\mathcal{I},:}$, $\mathbf{A}_{:,\mathcal{I}}$ and $\mathbf{A}_{\mathcal{I},\mathcal{I}}$ denote, respectively, the sub-matrices derived from matrix $\mathbf{A}$ by retaining respectively the rows, the columns and both at positions indexed by $\mathcal{I}$. After being processed by the weights matrix $\mathbf{M}^{j+1}$ of the next layer, the expression in (1) becomes

$$f^{j+1}(\mathbf{x}) = \left[\mathbf{M}^{j+1}_{:,\mathcal{I}}\mathbf{D}^j_{\mathcal{I},\mathcal{I}}\left[\mathbf{M}^j_{\mathcal{I},:}\mathbf{x} + \mathbf{b}^j_{\mathcal{I}}\right]_+ + \mathbf{b}^{j+1}_{\mathcal{I}}\right]_+. \tag{3}$$

In this sense, the sparsity of the diagonal of $\mathbf{D}^j$ encodes the computational complexity of the system, as it selects the effective dimensions of the $\mathbf{M}^j$ matrices and the $\mathbf{b}^j$ vectors.

3.2 Problem formulation

The architecture in Fig. 1 consists of a large number of parameters. Besides the variables $\mathbf{M}^j, \mathbf{D}^j, \mathbf{b}^j$ associates to each layer $j = 1, \ldots, J$ in (1), one needs to learn the vectors

$\mathbf{w}_1, \ldots, \mathbf{w}_K$ that define the SVM classifiers for the K classes. We will learn these variables from an annotated training set comprised of N training images $\mathbf{x}_i, i = 1, \ldots, N$, each with K labels $y_i^k \in \{-1, 1\}, k = 1, \ldots, K$ indicating whether image i belongs to class k or not. Given such a training set, our approach consists of minimizing the following objective over all the variables $\{(\mathbf{M}^j, \mathbf{b}^j, \mathbf{D}^j)\}_{j=1}^J$ and all the classifiers $\{\mathbf{w}^k\}_{k=1}^K$:

$$\frac{1}{K}\sum_{k=1}^{K}\left(\|\mathbf{w}^k\|_2^2 + \frac{C}{N}\sum_{i=1}^{N} l\left(y_i^k\,(f_J \circ \ldots \circ f_1(\mathbf{x}_i))^\top \mathbf{w}^k\right)\right) + \delta\sum_{j=1}^{J}\|\mathbf{D}^j\|_* + \mu\sum_{j=1}^{J}\|\mathbf{M}^j\|_F^2. \quad (4)$$

In the above expression, we have used *(i)* $l(x)$ to denote the hinge loss, given by $\max(0, 1-x)$; *(ii)* $\|\mathbf{D}\|_*$ to denote the trace norm, given by $\sum_i |D_{ii}|$ for the case of diagonal $\mathbf{D}$; and *(iii)* $\|\mathbf{M}\|_F^2$ to denote the squared Frobenius norm $\sum_{ij} M_{ij}^2$.

To illustrate the motivation behind this learning objective, we note first that the terms inside the summation over k in (4) are recognizable as an SVM objective for class k, where the scalar C is the SVM regularization parameter. The feature vectors used within this SVM objective are given by $f_J \circ \ldots \circ f_1(\mathbf{x}_i)$, which depends on $\{(\mathbf{M}^j, \mathbf{b}^j, \mathbf{D}^j)\}_{j=1}^J$. Hence we are learning the classifiers jointly with the feature extractor used to represent the input images. A similar approach has been used in [21] in the context of Fisher vector encoders to learn the encoder's GMM parameters under a discriminative objective.

The two regularization terms comprised of summations over j in (4) serve multiple purposes. One first purpose is to keep the SVM terms from decreasing indefinitely. Recall that the aim of an SVM objective is to maximize the margin between positive and negative examples. If it were not because of the penalty terms $\|\mathbf{w}^k\|_2^2$, it would be possible to minimize this indefinitely by multiplying the linear classifiers by a very large scalar. The penalty terms on the $\mathbf{D}^j$ and $\mathbf{M}^j$ serve a similar purpose when learning the features jointly with classifiers.

A second important purpose is to automatically select the shapes of the weights matrices $\mathbf{M}^j$ and $\mathbf{b}^j$. When applied to diagonal matrices such as $\mathbf{D}^j$, the trace norm is equivalent to an ℓ_1 norm computed from the diagonal vector of $\mathbf{D}^j$, and ℓ_1 norms are known to be well-behaved (*i.e.*, convex and differentiable almost everywhere) sparsity inducing norms. They are hence excellent surrogates for the ℓ_0 pseudo-norm that counts the number of non-zero entries of a vector. Since the matrices $\mathbf{D}^j$ multiply corresponding $\mathbf{M}^j$ and $\mathbf{M}^{j+1}$ matrices, low-valued coefficients of $\mathbf{D}^j$ can be compensated with increased norm of rows in $\mathbf{M}^j$ and columns in $\mathbf{M}^{j+1}$. The penalty terms on the norm of the $\mathbf{M}^j$'s hence address this ambiguity.

Approaches other than the one presented in (4) and Fig. 1 are indeed possible. For example, it would be possible to dispense altogether of the matrices $\mathbf{D}^j$ by using an alternative, structure inducing penalization on the matrices $\mathbf{M}^j$. The $\ell_{1,2}$ matrix norm is an appealing alternative. Letting $\mathbf{m}_i'$ denote the transposed i-th row of $\mathbf{M}$, it is given by

$$\|\mathbf{M}\|_{1,2} = \sum_i \|\mathbf{m}_i'\|_2, \quad (5)$$

and this is in turn an ℓ_1 penalization of the vector $[\|\mathbf{m}_i'\|_2]_i$ of ℓ_2 norms of the rows of $\mathbf{M}$. As such, it forces entire rows of $\mathbf{M}$ to be zero. Yet in the context of learning problems such as ours, Stochastic Gradient Descent (SGD) optimization methods that process a single example at a time are a must. An adaption of SGD is thus required that processes one example at a time while retaining nonetheless the sparse structure. As we will see next, our approach (4) using diagonal matrices can accomplish this with a simple adaptation of SGD-based solvers for ℓ_1 penalized SVM problems [1].

3.3 Learning approach

In order to derive an algorithm to minimize (4), we will first re-write it in a more convenient form using the following definitions:

$$r^{k,i} = \|\mathbf{w}^k\|_2^2 + Cl\left(y_i^k \left(f_J \circ \ldots \circ f_1(\mathbf{x})\right)^\top \mathbf{w}^k\right), \tag{6}$$

$$R_i = \frac{1}{K}\sum_{k=1}^{K} r^{k,i} + \delta \sum_{j=1}^{J} \|\mathbf{D}^j\|_* + \mu \sum_{j=1}^{J} \|\mathbf{M}^j\|_F^2, \tag{7}$$

the resulting form of (4) is

$$\frac{1}{N}\sum_{i=1}^{N} R_i. \tag{8}$$

In order to minimize (8), we will employ a block-coordinate SGD approach wherein, for each sample i, we process each block of coordinates $\mathbf{M}^1, \mathbf{D}^1, \mathbf{b}^1, \ldots, \mathbf{M}^J, \mathbf{D}^J, \mathbf{b}^J, \mathbf{w}^1, \ldots, \mathbf{w}^K$ successively. Letting θ denote a generic block of coordinates, the update step for that block at time instance t is as follows, where i_t is a sample index drawn randomly at time t, and the coefficient γ_t is the learning rate chosen using cross-validation:

$$\theta_{t+1} = \theta_t - \gamma_t \left.\frac{\partial R_{i_t}}{\partial \theta}\right|_{\theta_t}. \tag{9}$$

The required sub-gradient $\frac{\partial R_i}{\partial \theta}$ is given by

$$\frac{\partial R_i}{\partial \theta} = \frac{1}{K}\sum_{k=1}^{K} \frac{\partial r^{k,i}}{\partial \theta} + \delta \sum_{j=1}^{J} \frac{\partial \|\mathbf{D}^j\|_*}{\partial \theta} + \mu \sum_{j=1}^{J} \frac{\partial \|\mathbf{M}^j\|_F^2}{\partial \theta} \tag{10}$$

Expressions for $\frac{\partial r^{k,i}}{\partial \theta}$ when θ represents a classifier $\mathbf{w}^k$ are readily available from the literature on SGD-based SVM solvers [1, 13], and expressions for the case when θ represents a weights matrix $\mathbf{M}^j$ or offset vector $\mathbf{b}^j$ are available from the literature on deep learning [14]. The gradient of the squared Frobenius norm is just a rasterization of $2\mathbf{M}^j$ (whenever θ represents the corresponding block of coordinates $\mathbf{M}^j$).

Concerning the gradient with respect to the matrices $\mathbf{D}^j$, it is possible to apply the update rule specified in (10) directly, but this will produce matrices $\mathbf{D}^j$ with diagonals that are only approximately sparse. Instead, we will use a procedure adapted from ℓ_1 penalized SVM solvers [1]. To this end, we represent each $\mathbf{D}^j$ in terms of two non-negative vectors $\mathbf{v}^j$ and $\mathbf{u}^j$:

$$\mathbf{D}^j = \mathrm{diag}(\mathbf{v}^j) - \mathrm{diag}(\mathbf{u}^j), \text{ where } \mathbf{v}^j, \mathbf{u}^j \geq 0. \tag{11}$$

In order to enforce the non-negativeness of the $\mathbf{v}^j$ and $\mathbf{u}^j$, whenever we are optimizing over one of these blocks, we will follow the SGD update step in (9) by a projection into the set of non-negative vectors. The modified update step is given by

$$\theta_{t+1} = \left[\theta_t - \gamma_t \left.\frac{\partial R_{i_t}}{\partial \theta}\right|_{\theta_t}\right]_+. \tag{12}$$

Choice of learning rate. In order to make our algorithm converge at a fast rate, we will use an adaptive learning rate that gets updated and kept fixed for each batch of B samples. The learning rate is updated at the beginning of the batch using $\gamma_t = \gamma_{t-1} \cdot 2^{-n}$, where successive values of $n = 0, 1, 2, \ldots$ are tested until further increasing n no longer reduces the cost computed over a subset of the B samples from the batch. This approach has the advantage that the subset used to choose n is small, and hence adaptation is fast.

Early-stopping and choice of regularization parameters. We used two different early-stopping strategies that allowed us to select the number of iterations T (expressed as number of epochs) to use, both based on cross-validation. In one case we used the iteration that produced the highest mean Average Precision (mAP) over the validation set, while in the second case we chose the iteration giving the lowest cost, again computed over the validation set. We likewise use the validation set to choose the regularization parameters.

4 Results

In this section we evaluate our proposed learning algorithm and compare it against various state-of-the-art algorithms. To this end, we use the Pascal VOC 2007 dataset, which consists of 4192 test images and 5011 training images. We hold out 811 training images, choosing them uniformly over all classes, and use them as a validation set. As an image representation, we will use the VGG-M model of [2] which produces 128-dimensional features, accordingly using weights matrices of size 128×128.

Choice of regularization values Our learning algorithm is governed by three different regularization methods: the penalty weights μ and δ, and the number of training epochs T. The number of training epochs is determined using the validation set by computing either the *(i)* the validation mAP or *(ii)* the validation cost and choosing the number of training epochs that gives the best value. We found that using the validation mAP to determine T resulted in higher test mAP, but diagonal matrices $\mathbf{D}^j$ that are not necessarily sparse. Using the validation cost to determine T, on the other hand, resulted in slightly lower mAP values but in much higher sparsities for the matrices $\mathbf{D}^j$.

In Fig. 2 we illustrate the cross-validation method we use to choose the penalty weights μ and δ. The approach consists of keeping one penalty weight fixed while varying the second one, and then choosing the penalty weight that results in the highest validation mAP (this corresponded roughly to the value producing the highest test mAP).

Varying the number of layers J In Table 1, we evaluate the performance of our method as a function of the number of layers J in the architecture. We choose the regularization values following the procedure outlined above, using the validation cost to select the stopping point T. Note that our method succeeds in choosing sparse matrices $\mathbf{D}^j$ while retaining a high test mAP. For $J > 2$, only the first layer results in a matrix $\mathbf{D}^j$ with sparse diagonal. The reason for this is that the cross-validation procedure used to select δ in (7) only takes the mAP into account.

In Fig. 3, we hence plot both the sparsity for all layers and the corresponding test and validation mAPs when varying the penalty weight δ. Note that increasing δ drastically increases the number of zero diagonal entries in the architecture while only slightly affecting

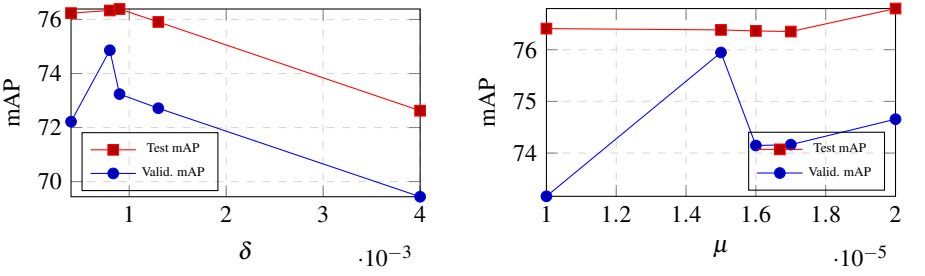

Figure 2: Effect of the penalty weights δ and μ for a single-layer architecture, using validation cost as a stopping criterion.

J	Test mAP	Number of zeros in diagonal of $\mathbf{D}^j$						δ	μ	T
		$j=1$	2	3	4	5	6			
1	76.38	93	-	-	-	-	-	8.0e-4	1.5e-5	120
2	76.20	105	105	-	-	-	-	1.4e-3	3.5e-5	180
3	76.58	92	0	0	-	-	-	8.0e-4	1.0e-5	240
5	76.55	91	0	0	0	0	-	8.0e-4	2.5e-5	360
6	76.19	88	0	0	0	0	0	7.0e-4	1.4e-5	420

Table 1: Using validation cost to choose the number of training epochs T, and validation mAP to choose the best δ first, and then the best μ.

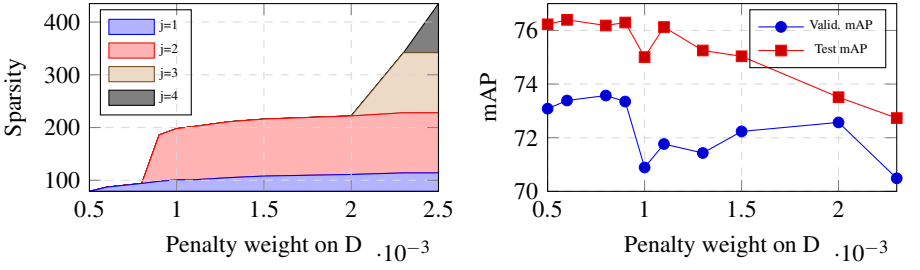

Figure 3: Effect of the penalty weight δ on *(left)* the number of zero diagonal entries of $\mathbf{D}^j$, $j=1,\ldots,4$, and *(right)* on the classification performance as measured by mAP. The zero diagonal entries are presented as stacked plots so that the vertical displacement of any shaded regions corresponds to the number of zero diagonal entries of $\mathbf{D}^j$ for the corresponding layer.

the classification performance. For example, for $\delta = 2.5e^{-3}$, close to 82% of the diagonal entries in all $\mathbf{D}^j$ are zero, while the test mAP has only dropped by 4.5%. For $\delta = 8e{-}4$, close to 40% of the diagonal entries are zero, while the system mAP is nearly unaffected.

For completeness, in Fig. 4 we present a plot of layer sparsity as a function of the training epoch. Note that $\mathbf{D}^1$ becomes sparse initially, and this reduces the intrinsic dimensionality of the signals at the input of the second layer, hence enabling $\mathbf{D}^2$ to become sparse.

Comparison against state-of-the-art In Table 2 we compare our proposed method against various state-of-the-art algorithms derived from CNN methods. We include four reference

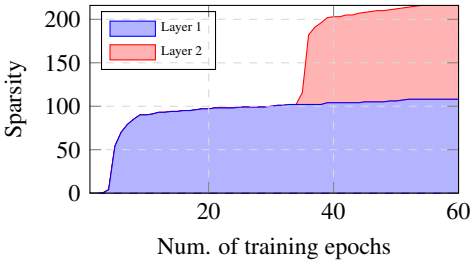

Figure 4: Number of zero entries in diagonal of $\mathbf{D}^j$ versus iteration number (expressed as number of epochs) for an architecture of $J = 2$ layers.

Method	Train time	Dim	# params.	mAP
CNN S TUNE-RNK_aug[2]	-	4K	∼100M	82.42
VGG-128_aug [2]	∼ 10s	128	2560	78.90
off-the-shelf_aug[17]	-	4K	81K	77.2
PRE1000C [15]	∼ 1 day	-	∼8.5M	77.73
VGG-128 [2]	∼ 10s	128	2560	76.34
off-the-shelf[17]	-	4K	81K	73.9
Ours($J = 1$)	210s	35	7040	76.38
Ours($J = 3$)	630s	36	9760	76.58
Ours($J = 6$)	1260s	128	100K	77.63

Table 2: Comparison of our proposed method with various existing CNN methods. The training time indicated includes only the training time related to Pascal VOC, and not training time incurred when learning on ImageNet. The top four methods rely on some form of data augmentation and have training sets that effectively many times bigger than the PascalVOC training set. The bottom five methods (including ours) only use the training images specified in PascalVOC.

methods that rely on data augmentation (the first four), and two reference methods that do not (the next two). The last three lines in the table correspond to three variants of our architecture. The first two have depth $J = 1,3$ and are learnt with sparsity in mind by using a stopping criterion T selected using validation cost. The last one is learned using validation mAP to select the stopping criterion T. Note that, for $J = 1,3$, our method produces very small features of size 35 and 36, respectively, while at the same time outperforming all the reference methods not-relying on augmentation, even when these use features many times larger. Our architecture with $J = 6$ results in a mAP value that outperforms not only the non-augmented references approaches, but also two of the four approaches relying on augmentation.

5 Conclusion

We present a method that automatically selects the size of the weight matrices inside fully-connected layers comprising a deep architecture. Our approach relies on a regularization

penalty term consisting of the ℓ_1 norm of the diagonal entries of diagonal matrices inserted between the fully-connected layers. Using such a penalty term forces the diagonal matrices to be sparse, accordingly selecting the effective number of rows and columns in the weights matrices of adjacent layers. We present a simple algorithm to solve the proposed formulation and demonstrate it experimentally on a standard image classification benchmark.

References

[1] Leon Bottou. Stochastic gradient descent tricks. In Grégoire Montavon, Geneviève Orr, and Klaus-Rober Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1. Springer, 2 edition, 2012. URL http://link.springer.com/chapter/10.1007/978-3-642-35289-8_25.

[2] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In *British Machine Vision Conference*, 2014. URL http://arxiv.org/abs/1405.3531.

[3] Mircea Cimpoi, Subhransu Maji, and Andrea Vedaldi. Deep convolutional filter banks for texture recognition and segmentation. *arXiv preprint arXiv:1411.6836*, 2014.

[4] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[5] Hanlin Goh, Nicolas Thome, Matthieu Cord, and Joo-Hwee Lim. Top-Down Regularization of Deep Belief Networks. In *Neural Information Processing Systems*, 2013.

[6] Yunchao Gong, Liwei Wang, Ruiqi Guo, and Svetlana Lazebnik. Multi-scale Orderless Pooling of Deep Convolutional Activation Features. In *European Conference on Computer Vision*, March 2014. URL http://arxiv.org/abs/1403.1840.

[7] Yunchao Gong, Liwei Wang, Ruiqi Guo, and Svetlana Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. *arXiv preprint arXiv:1403.1840*, 2014.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In *Pattern Analysis and Machine Intelligence*, 2015.

[9] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe : Convolutional Architecture for Fast Feature Embedding. In *ACM Multimedia*, 2014.

[10] Alex Krizhevsky, I. Sutskever, and Geoffrey Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Neural Information Processing Systems*, pages 1–9, 2012.

[11] Praveen Kulkarni, Joaquin Zepeda, Frederic Jurie, Patrick Perez, and Louis Chevallier. Hybrid Multi-Layer Deep CNN / Aggregator Feature for Image Classification. In *International Conference on Acoustics, Speech and Signal Processing*, 2015.

[12] Praveen Kulkarni, Joaquin Zepeda, Frédéric Jurie, Patrick Perez, and Louis Chevallier. Max-Margin, Single-Layer Adaptation of Transferred Image Features. In *BigVision Workshop, Computer Vision and Pattern Recognition*, 2015.

[13] Svetlana Lazebnik and Cordelia Schmid. Beyond Bags of Features : Spatial Pyramid Matching for Recognizing Natural Scene Categories. In *Computer Vision and Pattern Recognition*, 2006.

[14] Yann LeCun, Leon Bottou, Genevieve Orr, and Klaus-Robert Muller. Efficient BackProp. In *Neural Networks: Tricks of the Trade*, pages 9–50. 2002.

[15] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks. *Computer Vision and Pattern Recognition*, 2014.

[16] Florent Perronnin, J Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. *European Conference on Computer Vision*, 2010. URL http://link.springer.com/chapter/10.1007/978-3-642-15561-1_11.

[17] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN Features off-the-shelf : an Astounding Baseline for Recognition. In *Computer Vision and Pattern Recognition Workshops*, 2014.

[18] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos : Primal Estimated sub-GrAdient SOlver for SVM. In *International Conference of Machine Learning*, 2007.

[19] Josef Sivic and Andrew Zisserman. Video Google: A text retrieval approach to object matching in videos. In *International Conference on Computer Vision*, pages 2–9, 2003. ISBN 0769519504. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1238663.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. ISSN 15337928.

[21] Vladyslav Sydorov, Mayu Sakurada, and Christoph Lampert. Deep Fisher Kernels - End to End Learning of the Fisher Kernel GMM Parameters. In *Computer Vision and Pattern Recognition*, 2014. URL http://ist.ac.at/~chl/papers/sydorov-cvpr2014.pdf.

[22] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of Neural Networks using DropConnect. In *International Conference of Machine Learning*, 2013.