Matrix Backpropagation for Deep Networks with Structured Layers

Catalin Ionescu*[2,3], Orestis Vantzos†[3], and Cristian Sminchisescu‡[1,3]

[1]Department of Mathematics, Faculty of Engineering, Lund University
[2]Institute of Mathematics of the Romanian Academy
[3]Institute for Numerical Simulation, University of Bonn

Abstract

Deep neural network architectures have recently produced excellent results in a variety of areas in artificial intelligence and visual recognition, well surpassing traditional shallow architectures trained using hand-designed features. The power of deep networks stems both *from their ability to perform local computations followed by pointwise non-linearities over increasingly larger receptive fields,* and *from the simplicity and scalability of the gradient-descent training procedure based on backpropagation. An open problem is the inclusion of layers that perform global, structured matrix computations like segmentation (e.g. normalized cuts) or higher-order pooling (e.g. log-tangent space metrics defined over the manifold of symmetric positive definite matrices) while preserving the validity and efficiency of an end-to-end deep training framework. In this paper we propose a sound mathematical apparatus to formally integrate global structured computation into deep computation architectures. At the heart of our methodology is the development of the theory and practice of backpropagation that generalizes to the calculus of adjoint matrix variations. We perform segmentation experiments using the BSDS and MSCOCO benchmarks and demonstrate that deep networks relying on second-order pooling and normalized cuts layers, trained end-to-end using matrix backpropagation, outperform counterparts that do not take advantage of such global layers.*

1. Introduction

Recently, the end-to-end learning of deep architectures using stochastic gradient descent, based on very large datasets, has produced impressive results in realistic settings, for a variety of computer vision and machine learning domains[25, 37, 35]. There is now a renewed enthusiasm of creating integrated, automatic models that can handle the diverse tasks associated with an able perceiving system.

One of the most widely used architecture is the convolutional network (ConvNet) [26, 25], a deep processing model based on the composition of convolution and pooling with pointwise nonlinearities for efficient classification and learning. While ConvNets are sufficiently expressive for classification tasks, a comprehensive, deep architecture, that uniformly covers the types of non-linearities required for other visual calculations has not yet been established. In turn, matrix factorization plays a central role in classical (shallow) algorithms for many different computer vision and machine learning problems, such as image segmentation [36], feature extraction, descriptor design [18, 6], structure from motion [38], camera calibration [19], and dimensionality reduction [24, 4], among others. Singular value decomposition (SVD) in particular, is extremely popular because of its ability to efficiently produce global solutions to various problems.

In this paper we propose to enrich the dictionary of deep networks with layer generalizations and fundamental matrix function computational blocks that have proved successful and flexible over years in vision and learning models with global constraints. We consider layers that are explicitly *structure-aware* in the sense that they preserve global invariants of the underlying problem. Our paper makes two main mathematical contributions. The first shows how to operate with structured layers when learning a deep network. For this purpose we outline a *matrix generalization of backpropagation* that offers a rigorous, formal treatment of global properties. Our second contribution is to further derive and instantiate the methodology to learn convolutional networks for two different and very successful types of structured layers: 1) *second-order pooling* [6] and 2) *normalized cuts* [36]. An illustration of the resulting deep architecture for O_2P is given in fig. 1. In challenging datasets

*catalin.ionescu@ins.uni-bonn.de
†orestis.vantzos@ins.uni-bonn.de
‡cristian.sminchisescu@math.lth.se

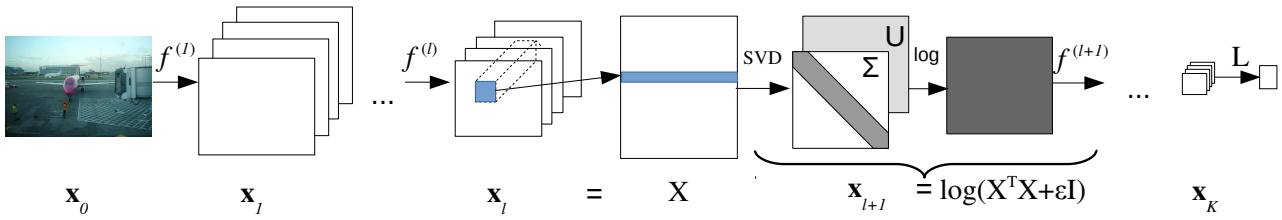

Figure 1. Overview of the DeepO$_2$P recognition architecture made possible by our methodology. The levels $1 \dots l$ represent standard convolutional layers. Layer $l+1$ is the global matrix logarithm layer presented in the paper. This is followed by fully connected layers and a logistic loss. The methodology presented in the paper enables analytic computation over both local and global layers, in a system that remains trainable end-to-end, for all its local and global parameters, using *matrix backpropagation*.

like BSDS and MSCOCO, we experimentally demonstrate the feasibility and added value of these two types of networks over counterparts that are not using global computational layers.

Related Work. Our work relates both to the extensive literature in the area of (deep) neural networks (see [35] for a review) with particular emphasis on ConvNets[25, 26] and with (shallow) architectures that have been proven popular and successful in computer vision[36, 3, 6, 8]. While deep neural networks models have focused, traditionally, on generality and scalability, the shallow computer vision architectures have often been designed with global computation and visual structure modeling in mind. Our objective in this work is to provide one possible approach towards formally marrying these two lines of work.

Bach and Jordan [3] introduced a (shallow) learning formulation for normalized cuts which we build upon in this work with several important differences. First, we aim to also learn the rank of the affinity matrix. Moreover while [3] aims to learn the parameters of their affinity model, we learn the feature representation of the data. Turaga *et al* [39] learn a model end-to-end based on a CNN by optimizing a standard segmentation criterion. While they used a simple connected component labeling as the grouping machinery, we demonstrate here the ability to group using more complex spectral techniques that are broadly applicable to a variety of problems.

Deep architectures have been recently designed for visual recognition by operating on top of figure-ground regions proposals from object segmentation[7, 40]. R-CNN [15] uses standard networks (e.g. AlexNet [25] or VGG-16 [37], which is an improved, deeper AlexNet). SDS [17] uses two AlexNet streams, one on the original image and the second one on the image with the background of the region masked. Relevant are also the architectures of He *et al* [20, 11], which use a global spatial pyramid pooling layer before the fully connected layers, in order to perform max-pooling over pyramid-structured image cells. Our architecture is also *complementary* to structured output formulations such as MRFs[5, 33, 8] which have been demonstrated to provide useful smoothing on top of high-performing CNN pixel classifier predictions [30]. Models developed at the same time with this work focus on the joint, end-to-end training of the deep feature extractor and the MRF[42, 9, 28]. We also show how a deep architecture can be trained jointly with structured layers, but in contrast focus on different models where global dependencies can be expressed as general matrix functions.

For recognition, we illustrate deep, fully trainable architectures, with a type of pooling layer that proved dominant for free-form region description [6], at the time on top of standard manually designed local features such as SIFT. In this context, our work is also related to kernel learning approaches over the manifold of positive-definite matrices [23]. However, we introduce different mathematical techniques related to matrix backpropagation, which has both the advantage of scalability and the one of learning compositional feature maps.

2. Deep Processing Networks

Let $\mathcal{D} = \{(\mathbf{d}^{(i)}, \mathbf{y}^{(i)})\}_{i=1\dots N}$ be a set of data points (e.g. images) and their corresponding desired targets (e.g. class labels) drawn from a distribution $p(\mathbf{d}, \mathbf{y})$. Let $L : \mathbb{R}^d \to \mathbb{R}$ be a *loss function* i.e. a penalty of mismatch between the *model prediction function* $f : \mathbb{R}^D \to \mathbb{R}^d$ with parameters W for the input $\mathbf{d}$ i.e. $f(\mathbf{d}^{(i)}, W)$ and the desired output $\mathbf{y}^{(i)}$. The foundation of many learning approaches, including the ones considered here, is the principle of *empirical risk minimization*, which states that under mild conditions, due to concentration of measure, the *empirical risk* $\hat{R}(W) = \frac{1}{N}\sum_{i=1}^{N} L(f(\mathbf{d}^{(i)}, W), \mathbf{y}^{(i)})$ converges to the true risk $R(W) = \int L(f(\mathbf{d}, W), \mathbf{y}) p(\mathbf{d}, \mathbf{y})$. This means it suffices to minimize the empirical risk to learn a function that will do well in general i.e.

$$\arg\min_{W} \frac{1}{N} \sum_{i=1}^{N} L(f(\mathbf{d}^{(i)}, W), \mathbf{y}^{(i)}) \qquad (1)$$

If L and f are both continuous (though not necessarily with continuous derivatives) one can use (sub-)gradient descent

on (1) for learning the parameters. This supports a general and effective framework for learning provided that a (sub-) gradient exists.

Deep networks, as a model, consider a class of functions f, which can be written as a series of successive function compositions $f = f^{(K)} \circ f^{(K-1)} \circ \ldots \circ f^{(1)}$ with parameter tuple $W = (\mathbf{w}_K, \mathbf{w}_{K-1}, \ldots, \mathbf{w}_1)$, where $f^{(l)}$ are called layers, $\mathbf{w}_l$ are the parameters of layer l and K is the number of layers. Denote by $L^{(l)} = L \circ f^{(K)} \circ \ldots \circ f^{(l)}$ the *loss as a function of the layer* $\mathbf{x}_{l-1}$. This notation is convenient because it conceptually separates the network architecture from the layer design.

Since the computation of the gradient is the only requirement for learning, an important step is the effective use of the principle of *backpropagation* (backprop). Backprop, as described in the literature, is an algorithm for efficiently computing the gradient of the loss with respect to the parameters. The algorithm recursively computes gradients with respect to both the inputs to the layers and their parameters (fig. 2) by making use of the *chain rule*. For a data tuple $(\mathbf{d}, \mathbf{y})$ and a layer l this is computing

$$\frac{\partial L^{(l)}(\mathbf{x}_{l-1}, \mathbf{y})}{\partial \mathbf{w}_l} = \frac{\partial L^{(l+1)}(\mathbf{x}_l, \mathbf{y})}{\partial \mathbf{x}_l} \frac{\partial f^{(l)}(\mathbf{x}_{l-1})}{\partial \mathbf{w}_l} \quad (2)$$

$$\frac{\partial L^{(l)}(\mathbf{x}_{l-1}, \mathbf{y})}{\partial \mathbf{x}_{l-1}} = \frac{\partial L^{(l+1)}(\mathbf{x}_l, \mathbf{y})}{\partial \mathbf{x}_l} \frac{\partial f^{(l)}(\mathbf{x}_{l-1})}{\partial \mathbf{x}_{l-1}} \quad (3)$$

where $\mathbf{x}_l = f^{(l)}(\mathbf{x}_{l-1})$ and $\mathbf{x}_0 = \mathbf{d}$ (data). The first expression is the gradient we seek (required for updating $\mathbf{w}_l$) whereas the second one is necessary for calculating the gradients in the layers below and updating their parameters.

3. Structured Layers

The existing literature concentrates on layers of the form $f^{(l)} = (f_1^{(l)}(\mathbf{x}_{l-1}), \ldots f_{d_{l+1}}^{(l)}(\mathbf{x}_{l-1}))$, where $f_j^{(l)} : \mathbb{R}^{d_l} \to \mathbb{R}$, thus $f^{(l)} : \mathbb{R}^{d_l} \to \mathbb{R}^{d_{l+1}}$. This simplifies processing significantly because in order to compute $\frac{\partial L^{(l)}(\mathbf{x}_{l-1}, \mathbf{y})}{\partial \mathbf{x}_{l-1}}$ there is a well defined notion of partial derivative with respect to the layer $\frac{\partial f^{(l)}(\mathbf{x}_{l-1})}{\partial \mathbf{x}_{l-1}}$ as well as a simple expression for the chain rule. However this formulation processes spatial coordinates independently and does not immediately generalize to more complex mathematical objects. Consider a matrix view of the (3-dimensional tensor) layer, $X = \mathbf{x}_{l-1}$, where $X_{ij} \in \mathbb{R}$, with i being the spatial coordinate[1] and j the index of the input feature. Then we can define a non-linearity on the entire $X \in \mathbb{R}^{m_l \times d_l}$, as a matrix, instead of each (group) of spatial coordinate separately. As the matrix derivative with respect to a vector (set aside to a matrix) is no longer well-defined, a matrix generalization of backpropation is necessary.

[1]For simplicity and without loss of generality we reshape (linearize) the tensor's spatial indices to one dimension with m_l coordinates.

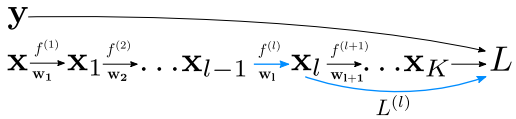

Figure 2. Deep architecture where data $\mathbf{x}$ and targets $\mathbf{y}$ are fed to a loss function L, via successively composed functions $f^{(l)}$ with parameters $\mathbf{w}_l$. *Backpropagation* (blue arrows) recursively expresses the partial derivative of the loss L w.r.t. the current layer parameters based on the partial derivatives of the next layer, *c.f.* eq. 2.

3.1. Computer Vision Models

To motivate the use of structured layers we will consider the following two models from computer vision:

1. *Second-Order Pooling* is one of the competitive hand-designed feature descriptors for regions [6] used in the top-performing method of the PASCAL VOC semantic segmentation, comp. 5 track [21, 27]. It represents global high-order statistics of local descriptors inside each region by computing a covariance matrix $X^\top X$ then applying a tangent space mapping [2] using the matrix logarithm, which can be computed using SVD. Instead of pooling over hand-designed local descriptors, such as SIFT [31], one could learn a ConvNet end-to-end, with a structured layer of the form

$$C = \log(X^\top X + \epsilon I) \quad (4)$$

where ϵI is a regularizer preventing $\log$ singularities around 0 when the covariance matrix is not full rank.

2. *Normalized Cuts* is an influential global image segmentation method based on pairwise similarities [36]. It constructs a matrix of local interactions e.g. $W = XX^\top$, then solves a generalized eigenvalue problem to determine a global image partitioning. Instead of manually designed affinities, one could, given a ground truth target segmentation, learn end-to-end the deep features that produce good normalized cuts.

3.2. Matrix Backpropagation

We call *matrix backpropagation* (MBP) the use of matrix calculus[12, 32, 14] to map between the partial derivatives $\frac{\partial L^{(l+1)}}{\partial \mathbf{x}_l}$ and $\frac{\partial L^{(l)}}{\partial \mathbf{x}_{l-1}}$ at two consecutive layers. We denote $X = \mathbf{x}_{l-1}$ and $Y = \mathbf{x}_l$. The processing can be performed in 2 steps[2]:

1. Derive $\mathcal{L}$, the functional describing the variations of the upper layer variables with respect to the lower layer

[2]The accompanying paper material[22] covers the background necessary to derive each of these steps.

variables.

$$dY = \mathcal{L}(dX) \tag{5}$$

This involves not only the forward mapping of the layer, $f^{(l)}$, but also the invariants associated to its variables. If X satisfies certain invariants, these need to be preserved to first (leading) order when computing dX. Invariants such as diagonality, symmetry, or orthogonality are explicitly enforced by our methodology.

2. Use the properties of the matrix inner product $A : B = \mathrm{Tr}(A^\top B)$ to obtain partial derivatives with respect to the lower layer variables. The usefulness of the colon-product comes from the Taylor expansion for functions $f : \mathbb{R}^{m\times n} \to \mathbb{R}$, which can be written as

$$f(X + dX) = f(X) + \frac{\partial f}{\partial X} : dX + \mathrm{O}(\|dX\|^2) \tag{6}$$

with $\left(\frac{\partial f}{\partial X}\right)_{ij} = \frac{\partial f}{\partial x_{ij}}$. This holds for a *general variation*, e.g. for a non-symmetric dX even if X itself is symmetric. To remain within a subspace like the one of symmetric, diagonal or orthogonal matrices, we can consider a projection of dX onto the space of admissible variations and then transfer the projection onto the derivative, to obtain the *projected gradient*. We use this technique repeatedly in our derivations.

Since the ":" operator is an inner product on the space of matrices, this is equivalent to constructively producing $\mathcal{L}^*$, a non-linear adjoint operator of $\mathcal{L}$

$$\frac{\partial L^{(l+1)}}{\partial Y} : dY = \frac{\partial L^{(l+1)}}{\partial Y} : \mathcal{L}(dX) \triangleq \mathcal{L}^*\left(\frac{\partial L^{(l+1)}}{\partial Y}\right) : dX$$
$$\Rightarrow \mathcal{L}^*\left(\frac{\partial L^{(l+1)}}{\partial Y}\right) = \frac{\partial L^{(l)}}{\partial X} \text{ by the chain rule} \tag{7}$$

The formulation is generally applicable to deep architectures with structured layers, where the processing can be expressed in matrix form.[3] Notice that matrix backpropagation cannot be derived using existing matrix 'cookbook' methodology[34] that compacts element-wise operations in matrix form. Our matrix calculus cannot be reduced to a sugar language constructed bottom-up from element-wise calculations. For intuition, consider global computations over arbitrary structured matrices based on eigen-decomposition. Recoursing to existing methodology, one would presumably express eigenvalues in closed-form and then work out their derivatives. However, no closed-form eigen-decomposition exists for matrices larger than 5×5 so that avenue is barred: one can not even write the layer forward step element-wise, let alone differentiate.

[3]Compared to an algorithmic or an index-by-index description of the layer (whenever that can be computed), our closed-form partial derivative calculations simplify both the implementation – since expressions are obtained via matrix operators – and the analysis. Numerical stability becomes important as the matrix decomposition functions that define the structured layers might be ill-conditioned, either intrinsically (e.g. SVD is not differentiable in the presence of repeated eigenvalues) or because of poorly chosen numerical schemes. This is true of both the forward and backward passes [16]. A methodology that provides a formal understanding of the underlying linear algebra, as given in this work, is thus essential.

4. Spectral and Non-Linear Layers

When global matrix operations are used in deep networks, they compound with other processing layers performed along the way. Such steps are architecture specific, although calculations like spectral decomposition are widespread, and central, in many vision and machine learning models. SVD possesses a powerful structure that allows one to express complex transformations like matrix functions and algorithms in a numerically stable form. In the sequel we show how the widely useful *singular value decomposition* (SVD) and the *symmetric eigenvalue problem* (EIG) can be leveraged towards constructing layers that perform global calculations in deep networks.

4.1. Spectral Layers

The first computational block we detail is matrix backpropagation for SVD problems. In the sequel, we denote $A_{sym} = \frac{1}{2}(A^\top + A)$ and A_{diag} be A with all off-diagonal elements set to 0.

Proposition 1 (SVD Variations) *Let $X = U\Sigma V^\top$ with $X \in \mathbb{R}^{m,n}$ and $m > n$, such that $U^\top U = I$, $V^\top V = I$ and Σ possessing diagonal structure. Then*

$$d\Sigma = (U^\top dX V)_{diag} \tag{8}$$

and

$$dV = 2V\left(K^\top \circ (\Sigma^\top U^\top dX V)_{sym}\right) \tag{9}$$

with

$$K_{ij} = \begin{cases} \frac{1}{\sigma_i^2 - \sigma_j^2}, & i \neq j \\ 0, & i = j \end{cases} \tag{10}$$

Consequently the partial derivatives are

$$\frac{\partial L}{\partial X} = U\left\{2\Sigma\left(K^\top \circ \left(V^\top \frac{\partial L}{\partial V}\right)\right)_{sym} + \left(\frac{\partial L}{\partial \Sigma}\right)_{diag}\right\} V^\top \tag{11}$$

Proposition 2 (EIG Variations) *Let $X = U\Sigma U^\top$ with $X \in \mathbb{R}^{m,m}$, such that $U^\top U = I$ and Σ possessing diagonal structure. Then the $d\Sigma$ is still* (8) *with $V = U$ but*

$$dU = 2U\left(K^\top \circ (U^\top dX U)_{sym}\right) \tag{12}$$

with

$$\tilde{K}_{ij} = \begin{cases} \frac{1}{\sigma_i - \sigma_j}, & i \neq j \\ 0, & i = j \end{cases} \tag{13}$$

The resulting partial derivatives are

$$\frac{\partial L}{\partial X} = U\left\{2\left(\tilde{K}^\top \circ \left(U^\top \frac{\partial L}{\partial V}\right)_{sym}\right) + \left(\frac{\partial L}{\partial \Sigma}\right)_{diag}\right\} U^\top \tag{14}$$

4.2. Non-Linear Layers

An (analytic) matrix function of a diagonalizable matrix $A = U\Sigma U^\top$ can be written as $f(A) = Uf(\Sigma)U^\top$. Since Σ is diagonal this is equivalent to applying f element-wise to Σ's diagonal elements. Combining this idea with the SVD decomposition $X = U\Sigma V^\top$, our matrix logarithm C can be written as an element-wise function $C = \log(X^\top X + \epsilon I) = V\log(\Sigma^\top\Sigma + \epsilon I)V^\top$.

The corresponding variations become

$$\begin{aligned} dC = &2\left(dV\log(\Sigma^\top\Sigma+\epsilon I)V^\top\right)_{sym} \\ &+ 2\left(V(\Sigma^\top\Sigma+\epsilon I)^{-1}\Sigma^\top d\Sigma V^\top\right)_{sym} \end{aligned}$$

Plugging it into $\frac{\partial L}{\partial C} : dC$ and manipulating the resulting formulas we obtain the partial derivatives

$$\frac{\partial L}{\partial V} = 2\left(\frac{\partial L}{\partial C}\right)_{sym} V\log(\Sigma^\top\Sigma+\epsilon I) \tag{15}$$

and

$$\frac{\partial L}{\partial \Sigma} = 2\Sigma(\Sigma^\top\Sigma+\epsilon I)^{-1}V^\top\left(\frac{\partial L}{\partial C}\right)_{sym} V \tag{16}$$

Plugging (16) and (15) into (11) gives the partial derivatives we seek.

5. Normalized Cut Layers

A central computer vision and machine problem is grouping (segmentation or clustering) i.e. discovering which datapoints (or pixels) belong to one of several partitions. A successful approach to clustering is *normalized cuts*. Let m be the number of pixels in the image and let $V = \{1,\ldots,m\}$ be the set of indices. We want to compute a partition $\mathcal{P} = \{P_1,\ldots,P_k\}$, where $k = |\mathcal{P}|$, $P_i \subset V$, $\bigcup_i P_i = V$ and $P_j \bigcap P_i = \varnothing$. This is equivalent to producing a matrix $E \in \{0,1\}^{m\times k}$ such that $E(i,j) = 1$ if $i \in P_j$ and 0 otherwise. Let $X \in \mathbb{R}^{m\times d}$ be a feature matrix with descriptor of size d and let W be a similarity matrix with positive entries. For simplicity we consider $W = X\Lambda X^\top$, where Λ is a $d\times d$ parameter matrix. Note that one can also apply other global non-linearities on top of the segmentation layer, as presented in the previous section. Let $D = [W\mathbf{1}]$, where $[\mathbf{v}]$ is the diagonal matrix with main diagonal $\mathbf{v}$, *i.e.* the diagonal elements of D are the sums of the corresponding rows of W. The *normalized cuts criterion* is

$$C(W,E) = \mathrm{Tr}(E^\top WE(E^\top DE)^{-1}) \tag{17}$$

Finding the matrix E that minimizes $C(W,E)$ is equivalent to finding a partitioning that minimizes the cut energy but penalizes unbalanced solutions.

It is easy to show[3] that $C(W,E) = k - \mathrm{Tr}(Y^\top D^{-1/2}WD^{-1/2}Y)$, where Y is such that a) $Y^\top Y = I$ and b) $D^{1/2}Y$ is piecewise constant with respect to E (i.e. it is equal to E times some scaling for each column). Ignoring the second condition we obtain a relaxed problem that can be solved, due to Ky Fan theorem, by an eigen-decomposition of $M = D^{-1/2}WD^{-1/2}$. [3] propose to learn the parameters Λ such that $D^{1/2}Y$ is piecewise constant because then solving the relaxed problem is equivalent to the original problem. In [3] the input features were fixed and therefore Λ are the only parameters used for alignment. This is not our case, as we place a global objective on top of convolutional network inputs. We can therefore leverage the network parameters in order to change X directly, thus training the bottom layers to produce a representation that is appropriate for normalized cuts.

To obtain a Y that is piecewise constant with respect to $D^{1/2}E$ we can align the span of M with that of $\Omega = D^{1/2}EE^\top D^{1/2}$. This can be achieved by minimizing the Frobenius norm of the corresponding projectors

$$J_1(W,E) = \frac{1}{2}\left\|\Pi_M - \Pi_\Omega\right\|_F^2 \tag{18}$$

We will obtain the partial derivatives of an objective with respect to the matrices it depends on, relying on matrix backpropagation to compute the derivatives.

Lemma 1 *Consider a symmetric matrix A and its orthogonal projection operator Π_A. If dA is a symmetric variation of A then*

$$d\Pi_A = 2\left((I-\Pi_A)dAA^+\right)_{sym} \tag{19}$$

The derivation, presented in full in our accompanying paper[22], relies only on basic properties of the projector with respect to itself and its matrix: $\Pi_A^2 = \Pi_A$ (idempotency of the projector) and $\Pi_A A = A$ (projector leaves the original space unchanged). Note that since $\Pi_A = AA^+$, there exists a spectral decomposition in training but it is hidden in A^+ (where A^+ is the Moore-Penrose inverse).

The resulting derivatives with respect to W are then

$$\begin{aligned} \frac{\partial J_1}{\partial W} = &D^{-1/2}\frac{\partial J_1}{\partial M}D^{-1/2} + \\ &\mathrm{diag}\left(D^{-1}\Omega\left(\frac{\partial J_1}{\partial \Omega}\right)_{sym} - D^{-1}M\left(\frac{\partial J_1}{\partial M}\right)_{sym}\right)\mathbf{1}^\top \end{aligned}$$

A related optimization objective is

$$J_2 = \frac{1}{2}\left\|\Pi_W - \Pi_\Psi\right\|_F^2, \tag{20}$$

with $\Psi = E(E^\top E)^{-1}E^\top$. Its partial derivative can be written as

$$\frac{\partial J_2}{\partial W} = -2(I - \Pi_W)\Pi_\Psi W^+ \tag{21}$$

Finally, propagating the partial derivatives of J_i down to Λ and X gives

$$\frac{\partial J_i}{\partial \Lambda} = X^\top \frac{\partial J_i}{\partial W} X \tag{22}$$

and

$$\frac{\partial J_i}{\partial X} = 2\left(\frac{\partial J_i}{\partial W}\right)_{sym} X\Lambda^\top \tag{23}$$

An important aspect of our approach is that we do not restrict the rank in training. During alignment the optimization may choose to collapse certain directions thus reducing rank. We prove a topological lemma [22] implying that if the Frobenius distance between the projectors (such as in the two objectives J_1, J_2) drops below a certain value, then the ranks of the two projectors will match. Conversely, if for some reason the ranks cannot converge, the objectives are bounded away from zero.

6. Experiments

In this section we validate the proposed methodology by constructing models on standard datasets for region-based object classification, like Microsoft COCO [29], and for image segmentation on BSDS [1]. A matconvnet [41] implementation of our models and methods is publicly available.

6.1. Region Classification on MSCOCO

For recognition we use the MSCOCO dataset [29], which provides 880k segmented training instances across 80 classes, divided into training and validation sets. The main goal is to assess our second-order pooling layer in various training settings. A secondary goal is to study the behavior of ConvNets learned from scratch on segmented training data. This has not been explored before in the context of deep learning because of the relatively small size of the datasets with associated object segmentations, such as PASCAL VOC [13].

The experiments in this section use the convolutional architecture component of AlexNet[25] with the global O_2P layers we propose in order to obtain DeepO_2P models with both classification and fully connected (FC) layers in the same topology as Alexnet. We crop and resize each object bounding box to have 200 pixels on the largest side, then pad it to the standard AlexNet input size of 227x227 with small translation jittering, to limit over-fitting. We also randomly flip the images in each mini-batch horizontally, as in standard practice. Training is performed with stochastic gradient descent with momentum. We use the same batch size (100 images) for all methods but the learning rate was optimized for each model independently. All the DeepO_2P models used the same $\epsilon = 10^{-3}$ parameter value in (4).

Architecture and Implementation details. Implementing the spectral layers efficiently is challenging since the GPU support for SVD is still very limited and our parallelization efforts even using the latest CUDA 7.0 solver API have delivered a slower implementation than the standard CPU-based. Consequently, we use CPU implementations and incur a penalty for moving data back and forth to the CPU. The numerical experiments revealed that an implementation in single precision obtained a significantly less accurate gradient for learning. Therefore all computations in our proposed layers, both in the forward and backward passes, are in double precision. In experiments we still noticed a significant accuracy penalty due to inferior precision in all the other layers (above and below the structured ones), still computed in single precision, on the GPU.

In our extend paper[22] we present a second formal derivation of the non-linear spectral layer based on eigendecomposition of $Z = X^\top X + \epsilon I$ instead of SVD of X. Our experiments favor the derivation presented here. The alternative implementation, which is formally correct, exhibits numerical instability in the derivative when multiple eigenvalues have very close values, thus producing blow up in $\tilde{K}$. Such numerical issues are expected to appear under some implementations, when complex layers like the ones presented here are integrated in deep network settings.

Results. The results of the recognition experiment are presented in table 1. They show that our proposed DeepO_2P-FC models, containing global layers, outperform standard convolutional pipelines based on AlexNet, on this problem. The bottom layers are pre-trained on ImageNet using AlexNet, and this might not provide the ideal initial input features. However, despite this potentially unfavorable initialization, our model jointly refines all parameters (both convolutional, and corresponding to global layers), end to end, using a consistent cost function.

We note that the fully connected layers on top of the DeepO_2P layer offer good performance benefits. O_2P over hand-crafted SIFT performs considerably less well than our DeepO_2P models, suggesting that large potential gains can be achieved when deep features replace existing descriptors.

6.2. Full-Image Segmentation on BSDS300

We use the BSDS300 dataset to validate our deep normalized cuts approach. BSDS contains 200 training images and 100 testing images and human annotations of all the relevant regions in the image. Although small by the standards of neural network learning it provides exactly the supervision we need to refine our model using global information. Note that since the supervision is pixel-wise, the number of effective datapoint constraints is much larger. We evaluate using the average and best covering metric under the Opti-

Method	SIFT-O_2P	AlexNet	AlexNet (S)	DeepO_2P	DeepO_2P(S)	DeepO_2P-FC	DeepO_2P-FC(S)
Results	36.4	25.3	27.2	28.6	32.4	**25.2**	28.9

Table 1. Classification error on the validation set of MSCOCO (lower is better). Models with (S) suffixes were trained from scratch (i.e. random initialization) on the MSCOCO dataset. The DeepO_2P models only use a classification layer on top of the DeepO_2P layer whereas the DeepO_2P-FC also have fully connected layers in the same topology as AlexNet. All parameters of our proposed global models are refined jointly, end-to-end, using the proposed matrix backpropagation.

Arch	[10]	AlexNet		VGG			
Layer		ReLU-5		ReLU-4		ReLU-5	
Method	NCuts	NCuts	DeepNCuts	NCuts	DeepNCuts	NCuts	DeepNCuts
Results	.55 (.44)	.59 (.49)	.65 (.56)	.65 (.56)	.73 (.62)	.70 (.58)	.74 (.63)

Table 2. Segmentation results give best and average covering to the pool of ground truth segmentations on the BSDS300 dataset [1] (larger is better). We use as baselines the original normalized cuts [10] using intervening contour affinities as well as normalized cuts with affinities derived from non-finetuned deep features in different layers of AlexNet (ReLU-5 - the last local ReLU before the fully connected layers) and VGG (first layer in block 4 and the last one in block 5). Our DeepNCuts models are trained end-to-end, based on the proposed matrix backpropagation methodology, using the objective J_2.

mal Image Scale (OIS) criterion [1]. Given a set of full image segmentations computed for an image, this amounts to selecting the one that maximizes the average and best covering, respectively, compared to the pool of ground truth segmentations.

Architecture and Implementation details. We use both the AlexNet[25] and the VGG-16[37] architectures to feed our global layers. All the parameters of the deep global models (including the low-level features, pretrained on ImageNet) are refined end-to-end using the J_2 objective. We use a linear affinity but we need all entries of W to be positive. Thus, we use ReLU layers to feed the segmentation ones. Initially, we just cascaded our segmentation layer to different layers in AlexNet but the resulting models were hard to learn. Our best results were obtained by adding two Conv-ReLU pairs initialized randomly before the normalized cuts layer. This results in many filters in the lower layer (256 for AlexNet and 1024 for VGG) for high capacity but few in the top layer (20 dimensions) to limit the maximal rank of W. For AlexNet we chose the last convolutional layer while for VGG we used both the first ReLU layer in block[4] 4 and the top layer from block 5. This gives us feeds from layers with different invariances, receptive field sizes (32 vs. 132 pixels) and coarseness (block 4 has $2\times$ the resolution of 5). We used an initial learning rate of 10^{-4} but $10\times$ larger rates for the new layers. A dropout layer with rate .25, between the last two layers, reduces overfitting. In inference, we generate 8 segmentations by clustering[3] and split connected components into separate segments.

Results. The results in table 2 show that in all cases we obtain important performance improvements with respect to the corresponding models that perform inference directly on original AlexNet/VGG features. Training using our Matlab implementation takes 2 images/s considering 1 image per batch while testing at about 3 images/s on a standard Titan Z GPU with an 8 core E5506 CPU. In experiments we monitor both the objective and the rank of the similarity matrix. Rank reduction is usually a good indicator of performance in both training and testing. In the context of the rank analysis in §5, we interpret these findings to mean that if the rank of the similarity is too large compared to the target, the objective is not sufficient to lead to rank reduction. However if the rank of the predicted similarity and the ground truth are initially not too far apart, then rank reduction (although not always rank matching) does occur and improves the results.

[4]We call a block the set of layers between two pooling levels.

7. Conclusion

Motivated by the recent success of deep network architectures, in this work we have introduced the mathematical theory and the computational blocks that support the development of more complex models with layers that perform structured, global matrix computations like segmentation or higher-order pooling. Central to our methodology is the development of the matrix backpropagation methodology which relies on the calculus of adjoint matrix variations. We provide detailed derivations (see also our extended version[22]), operating conditions for spectral and non-linear layers, and illustrate the methodology for normalized cuts and second-order pooling layers. Our extensive region recognition and segmentation experiments based on MSCoco and BSDS show that that deep networks relying on second-order pooling and normalized cuts layers, trained end-to-end using the introduced practice of matrix backpropagation, outperform counterparts that do not take advantage of such global layers.

Acknowledgements. This work was partly supported by CNCS-UEFISCDI under CT-ERC-2012-1, PCE-2011-3-0438, JRP-RO-FR-2014-16. We thank J. Carreira for helpful discussions and Nvidia for a graphics board donation.

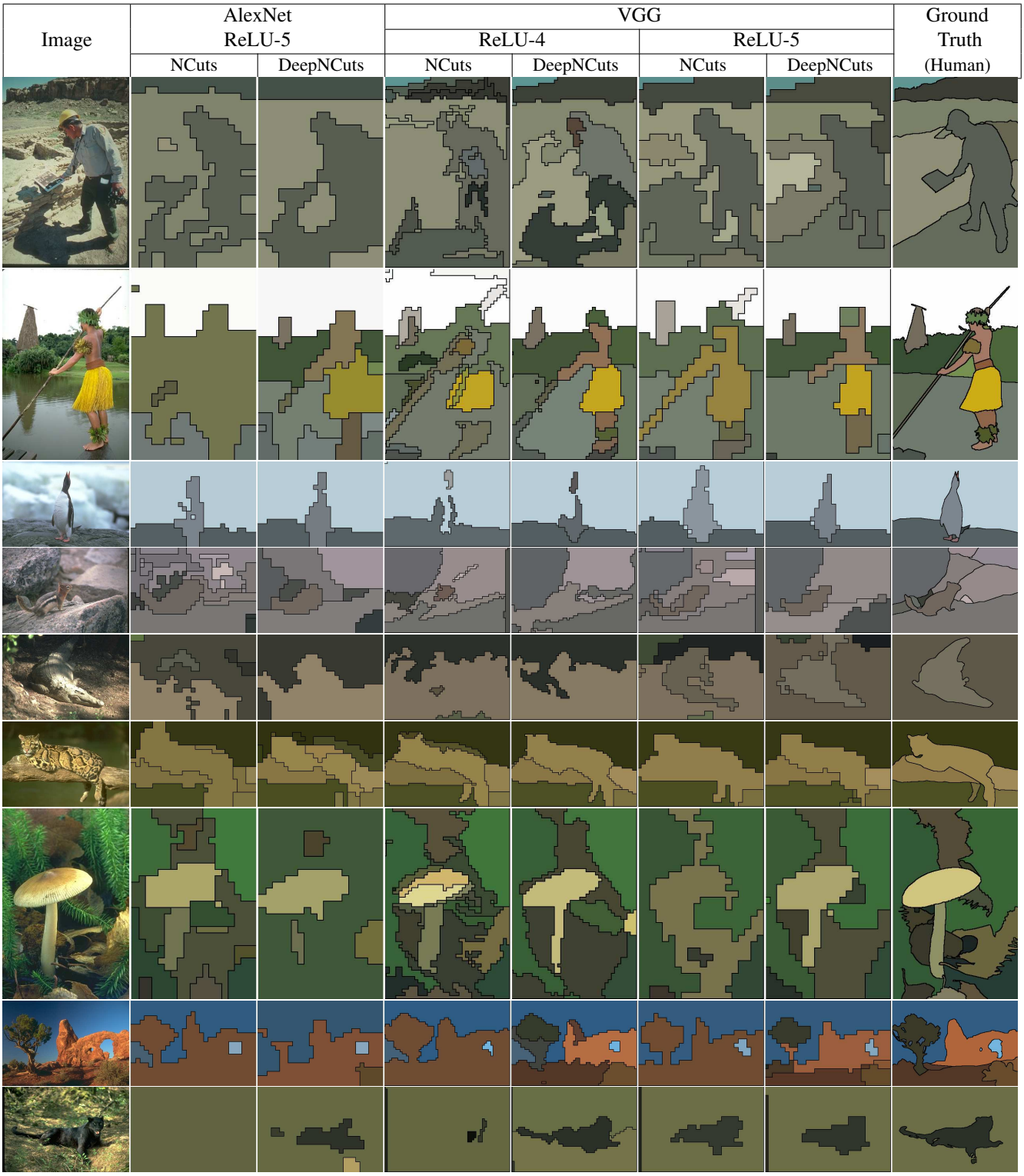

Figure 3. Segmentation results on images from the test set of BSDS300. We show on the first column the input image followed by a baseline (original parameters) and our DeepNcuts both using AlexNet ReLU-5. Two other pairs of baselines and DeepNCut models trained based on the J_2 objective follow. The first pair uses ReLU-4 and the second ReLU-5. The improvements obtained by learning are both quantitatively significant and easily visible on this side-by-side comparison.

References

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *PAMI*, 33(5):898–916, May 2011.

[2] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache. Geometric means in a novel vector space structure on symmetric positivedefinite matrices. *SIAM Journal on Matrix Analysis and Applications*, 29(1):328–347, 2007.

[3] F. R. Bach and M. I. Jordan. Learning spectral clustering, with application to speech separation. *JMLR*, 7:1963–2001, 2006.

[4] M. Belkin and P. Niyogi. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation*, 15(6):1373–1396, 2003.

[5] L. Bottou, Y. Bengio, and Y. Le Cun. Global training of document processing systems using graph transformer networks. In *CVPR*, pages 489–494. IEEE, 1997.

[6] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *ECCV*, pages 430–443. Springer, 2012.

[7] J. Carreira and C. Sminchisescu. CPMC: Automatic Object Segmentation Using Constrained Parametric Min-Cuts. *PAMI*, 2012.

[8] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015.

[9] L. C. Chen, A. G. Schwing, A. L. Yuille, and R. Urtasun. Learning deep structured models. In *ICML*, 2015.

[10] T. Cour, F. Benezit, and J. Shi. Spectral segmentation with multiscale graph decomposition. In *CVPR*, 2005.

[11] J. Dai, K. He, and J. Sun. Convolutional feature masking for joint object and stuff segmentation. In *CVPR*, 2015.

[12] P. S. Dwyer and M. MacPhail. Symbolic matrix derivatives. *The Annals of Mathematical Statistics*, pages 517–534, 1948.

[13] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The Pascal visual object classes (VOC) challenge. *IJCV*, 88(2):303–338, 2010.

[14] M. B. Giles. Collected matrix derivative results for forward and reverse mode algorithmic differentiation. In *Advances in Automatic Differentiation*, pages 35–44. Springer, 2008.

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587. IEEE, 2014.

[16] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, USA, 1996.

[17] B. Hariharan, P. Arbelaez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *ECCV*. 2014.

[18] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[19] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *PAMI*, 2015.

[21] A. Ion, J. Carreira, and C. Sminchisescu. Probabilistic Joint Image Segmentation and Labeling. In *Advances in Neural Information Processing Systems*, December 2011.

[22] C. Ionescu, O. Vantzos, and C. Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *CoRR*, abs/1509.07838, 2015.

[23] S. Jayasumana, R. Hartley, M. Salzmann, H. Li, and M. Harandi. Kernel methods on the riemannian manifold of symmetric positive definite matrices. In *CVPR*. IEEE, 2013.

[24] I. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, 2002.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] F. Li, J. Carreira, G. Lebanon, and C. Sminchisescu. Composite statistical inference for semantic segmentation. In *CVPR*, pages 3302–3309. IEEE, 2013.

[28] G. Lin, C. Shen, I. D. Reid, and A. v. d. Hengel. Efficient piecewise training of deep structured models for semantic segmentation. *CoRR*, abs/1504.01013, 2015.

[29] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollr, and C. Zitnick. Microsoft COCO: Common Objects in Context. In *ECCV*, 2014.

[30] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. In *CVPR*, 2015.

[31] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, volume 2, pages 1150–1157. Ieee, 1999.

[32] J. R. Magnus and H. Neudecker. *Matrix differential calculus with applications in statistics and econometrics*. J. Wiley & Sons, Chichester, New York, Weinheim, 1999.

[33] J. Peng, L. Bo, and J. Xu. Conditional neural fields. In *NIPS*, pages 1419–1427, 2009.

[34] K. B. Petersen and M. S. Pedersen. The matrix cookbook, nov 2012. Version 20121115.

[35] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015.

[36] J. Shi and J. Malik. Normalized cuts and image segmentation. *PAMI*, 22(8):888–905, Aug 2000.

[37] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[38] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: a factorization method. *IJCV*, 9(2):137–154, 1992.

[39] S. Turaga, K. Briggman, M. N. Helmstaedter, W. Denk, and S. Seung. Maximin affinity learning of image segmentation. In *NIPS*, pages 1865–1873, 2009.

[40] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *IJCV*, 2013.

[41] A. Vedaldi and K. Lenc. MatConvNet – Convolutional Neural Networks for MATLAB. In *Int. Conf. Multimedia*, 2015.

[42] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr. Conditional random fields as recurrent neural networks. *ICCV*, 2015.