Coarse-to-Fine Deep Kernel Networks

Hichem Sahbi
CNRS, UPMC
Sorbonne Universités, Paris
hichem.sahbi@lip6.fr

Abstract

In this paper, we address the issue of efficient computation in deep kernel networks. We propose a novel framework that reduces dramatically the complexity of evaluating these deep kernels. Our method is based on a coarse-to-fine cascade of networks designed for efficient computation; early stages of the cascade are cheap and reject many patterns efficiently while deep stages are more expensive and accurate. The design principle of these reduced complexity networks is based on a variant of the cross-entropy criterion that reduces the complexity of the networks in the cascade while preserving all the positive responses of the original kernel network. Experiments conducted – on the challenging and time demanding change detection task, on very large satellite images – show that our proposed coarse-to-fine approach is effective and highly efficient.

1. Introduction

With the era of big data, there is an exponential growth of image collections in the web and this makes their manual annotation and search completely out of reach. With this growth rate, there is an urgent need for reliable and also efficient automatic solutions able to annotate and search these large collections. Visual concept detection is one of these major challenges that consists of recognizing and localizing concepts/events into flows of visual contents using variety of machine learning and inference techniques; among these techniques, deep and convolutional neural networks [15, 11] are particularly successful (see for instance [29, 20, 41, 43, 14, 34, 19, 22, 27, 50, 21]). Recent breakthroughs and success stories of deep learning – in vision, pattern recognition and neighboring fields – are also due to the development of extremely efficient hardware resources that make running deep learning models on bigdata much more tractable. However, on widely used cheap hardware devices, deep learning models are still very time demanding and require careful algorithmic design in order to achieve efficient computation while maintaining a high accuracy.

Deep learning models usually operate on vectorial data; the underlying parametric models take vectorial inputs and return discriminatively-trained representations and similarities [9, 44]. When only relationships (or similarities) between input data are available[1], deep kernel networks become better alternatives [42, 51, 25, 26, 47, 8]. These networks are defined as recursive multi-layered combinations of standard kernels (e.g., Gaussian, random walks, etc.) that capture simple linear as well as intricate nonlinear relationships between input data. Learning the parameters of these networks together with classifiers allows us to achieve deep learning on non-vectorial data[2] more effectively compared to existing standard kernels as well as shallow multiple kernels [10, 3]. However, the downside of the deep kernel networks resides in their computational overhead. Indeed, the computational complexity of evaluating these networks scales quadratically w.r.t the cardinality of data and this is further exaggerated in the regime of very deep networks. Existing state-of-the art solutions mitigate this issue; for instance, authors in [5] reduce the number of kernel evaluations by approximating a heavy kernel-based radial-basis function with a reduced set of kernels, and authors in [32] train convolutional networks that best capture a particular class of invariant kernels. Other generic solutions reduce the number of units and connection weights (and hence speedup deepnets) using pruning and weight sharing [17, 17, 49, 18], singular value decomposition [12], regularization and sparsity [46, 7, 39, 48] as well as hardware design [16]. Our proposed solution, in this paper, is conceptually different from all the aforementioned techniques: on the one hand, our efficient kernel network design is not restricted to a specific class of kernels; it is generic and can be applied to more general classes of deep kernels (see also Eq. 1 and Section 3). On the other hand, in sparsity and regularization based methods, a given (targeted) cost may not necessarily be reached (i.e., after solving optimization) while in our

[1] for instance through graphs in social networks.
[2] in the Hilbert space associated to these kernels.

method any targeted cost, fixed a priori, can be reached (at the expense of a some loss in precision) thanks to the coarse-to-fine design as shown subsequently.

In this paper, we propose a novel coarse-to-fine framework for efficient deep kernel network evaluation. This approach is based on a cascade of kernel networks with a gradual increase of complexity and discrimination power. Networks in the early stages of the cascade are relatively shallow and used to reject most of the dominant patterns (with a negligible cost) while networks in the subsequent stages of the cascade are more accurate (but expensive) and reserve intense computation only to the rare positive patterns. This makes the cascade very suitable for classification problems – such as *change detection* in very large satellite images – where "target/no-target" classes are very imbalanced.

Starting from a pretrained deep kernel network (referred to as f-network) which is also highly accurate and expensive, we build its surrogates (referred to as g-networks) with a reduced complexity using a variant of the cross-entropy criterion. The latter minimizes the differences between the outputs of classifiers trained on top of the f and g-networks. Note that the complexity of the g-networks (measured by the number of units and depth) is fixed a priori depending on the expected amount of computation that makes the overall evaluation cost of the cascade cheap (see also Sections 3, 4). As the g-networks are naturally rank-deficient (i.e., their error rates are intrinsically higher than the f-networks), these g-networks are designed in order to satisfy the *conservation hypothesis*: the latter states that all the positive responses of classifiers built on top of the f-network should be preserved by classifiers built on top of the g-networks. We implement this hypothesis using a particular weighting scheme (of the cross-entropy criterion) that favors very small false negative rates to the detriment of an increase of false alarms. In spite of this increase of false alarms, most of these alarms (dominant patterns) are rejected in the early stages of the cascade and only a small fraction requires further processing using more expensive and accurate networks in the subsequent stages. Note that our coarse-to-fine processing belongs to the "ϵ-lossy" approaches that have been successfully applied to popular problems such as face detection using hierarchies of classifiers [13, 1, 2, 30]. To the best of our knowledge, none of these solutions tackled the issue of speeding-up deep kernel networks and most of the existing solutions were dedicated to support vector machines [28, 37, 38, 36, 33] and boosting [45, 31].

The remainder of this paper is organized as follows; section 2 provides details about deep kernel networks while section 3 introduces the main contribution; a coarse-to-fine approach that reduces the computational complexity of these networks. Section 4 shows the efficiency and the effectiveness of our method on the challenging problem of change detection in large and high resolution satellite imagery. Finally, section 5 concludes the paper while providing possible extensions for a future work.

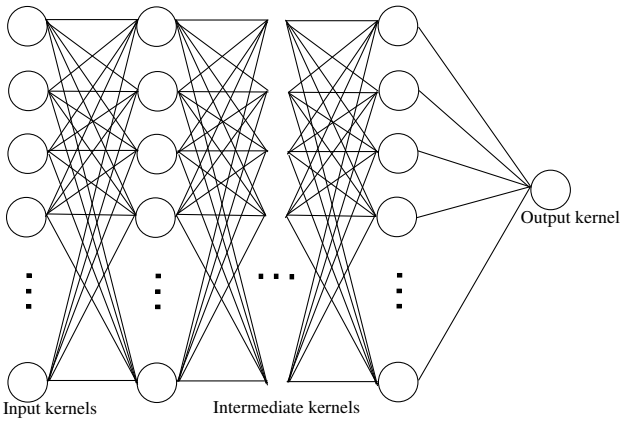

Figure 1. This figure shows an example of a deep kernel network.

2. Deep Kernel Networks

Consider a collection of ℓ labeled training samples $\mathcal{L} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{\ell}$, with $\mathbf{x}_i \in \mathbb{R}^d$ being a feature vector (for instance the VGG-net descriptor [40]) and $\mathbf{y}_i$ its class label in $\{-1, +1\}$ and another collection of u test samples $\mathcal{U} = \{\mathbf{x}_i\}_{i=\ell+1}^{\ell+u}$. Our goal is to jointly learn a deep multiple kernel and classifier f from the labeled samples; here f is an SVM-based classifier that predicts the class $\mathbf{y}_i \leftarrow \text{sign}[f(\mathbf{x}_i)]$ of a given sample $\mathbf{x}_i \in \mathcal{U}$. Usual algorithms, such as shallow multiple kernel learning (MKL) [3], jointly learn kernels and SVM-based classifiers by maximizing a margin using an EM-like optimization; as shown through this work, we consider instead a deep version of MKL which is highly effective and efficient.

2.1. Deep Multiple Kernels

A kernel (denoted as κ) is a symmetric function that provides a similarity between any two given samples [10]. When positive semi-definite, κ can be written as an inner product in a high (possibly infinite) dimensional space, via a mapping function (denoted as ϕ). Among the existing kernels, polynomial and radial basis functions are the most studied [10]. In this work, we aim to learn an implicit mapping function that recursively characterizes a nonlinear and deep combination of multiple existing kernels.

Fig. 1 shows our deep kernel network with L layers. For each layer l and its associated unit p, a kernel domain $\big\{\kappa_p^{(l)}(\cdot,\cdot)\big\}$ is recursively defined as

$$\kappa_p^{(l)}(\cdot,\cdot) = h\big(\sum_q \mathbf{w}_{q,p}^{(l-1)}\ \kappa_q^{(l-1)}(\cdot,\cdot)\big), \qquad (1)$$

where h is a nonlinear activation function[3]. In the above equation, $q \in \{1, \ldots, n_{l-1}\}$, n_{l-1} is the number of units in layer $(l-1)$ and $\{\mathbf{w}_{q,p}^{(l-1)}\}_q$ are the (learned) weights associated to kernel $\kappa_p^{(l)}$. In particular, $\{\kappa_p^{(1)}\}_p$ are the input kernels including Gaussian, etc. When $L = 2$, the architecture is shallow, and it is equivalent to the nonlinear version of MKL (see for instance Zhuang et al. [51]). For larger values of L, the network becomes deep.

For any given pair of samples, a vector containing the values of different standard (elementary) kernels on this pair is evaluated and considered as an input to our deep network. These elementary kernel values are then forwarded to the subsequent intermediate layer resulting into n_2 multiple kernels through the nonlinear combination of the previous layer, etc. The final kernel is a highly nonlinear combination of elementary kernels.

Note that with this setting, deep kernel network evaluation is inductive, and the computation feasible on any new pairs of samples. Note also that the deep kernel network in essence is a multi-layer perceptron (MLP), with nonlinear activation functions. The difference is that the last layer is not designed for classification, rather than to deliver a similarity value. However, we can use the classical backpropagation algorithm specific for MLP to optimize the weights in the deep kernel network. Let J denotes an objective function associated to our classification problem. More details, about choice of J, are discussed in Section 3. We assume that the computation of gradients of the objective function J w.r.t the output kernel $\kappa_1^{(L)}$ (i.e. $\frac{\partial J}{\partial \kappa_1^{(L)}(.,.)}$) is tractable. According to the chain rule, the corresponding gradients w.r.t coefficients $\mathbf{w}$ are computed, and then used to update these weights using gradient descent.

3. Coarse-to-Fine Deep Kernel Networks

Let f be an SVM classifier trained on top of the deep kernel $\kappa_1^{(L)}$; in what follows, $\kappa_1^{(L)}$ is simply rewritten as κ_f. In practice, the depth of this deep kernel network (and also the number of its units) should be sufficiently large in order to optimize the generalization performance of f (see [25, 23, 24]). However, deep kernel networks may affect the computational efficiency of f as the evaluation cost of the underlying deep kernel κ_f becomes extremely prohibitive; particularly on limited hardware resources.

Our goal is to make the evaluation cost of these kernel networks cheap by reducing their complexity while maintaining their high accuracy. As shown through this paper, this is achieved using a well optimized cascade of deep networks (and classifiers) that quickly rejects simple patterns which belong to the *dominant* class while reserving intense computation only to the *rare* targeted class (see Fig. 2).

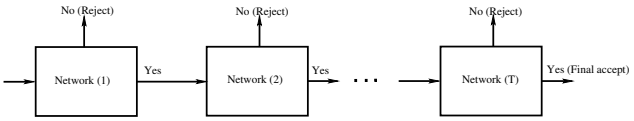

Figure 2. This figure shows the cascade of T classifiers and g-(kernel)-networks (as shown in experiments, T is set to 6).

3.1. The f-network vs. the g-networks

In what follows, the f-network refers to the original deep kernel network while its g-variant corresponds to its simplified version. Considering a pretrained classifier f (and its associated f-network), building a single monolithic classifier g (on top of a reduced complexity g-network) – which is strictly equivalent to f – is clearly out of reach; at least because the representational power[4] of the f-network is higher than the g-network.

In order reduce the computational complexity of evaluating f, we proceed differently. We consider a *coarse-to-fine* cascade of classifiers $\{g_t\}_{t=1}^T$; any given g_t is a simplified instance of f. More precisely, $\{g_t\}_t$ are associated to deep kernel networks with increasing complexities (measured by the depth and number of units). Test patterns are fed to this cascade and classified in a coarse-to-fine way; a test pattern $\mathbf{x}$ is declared as positive iff all the classifiers $\{g_t\}_{t=1}^T$ answer positively, while $\mathbf{x}$ is quickly rejected (as negative) if one of the classifiers $\{g_t\}_t$ answers negatively. As the negative class is dominant, the overall evaluation cost of the cascade is dominated by the cost to reject the negative patterns. Hence, the high efficiency of the cascade is dependent on the ability of the classifiers (in the early stages) to reject negative patterns quickly while maintaining the positive scores of the f-classifier (i.e., classifier at the final stage). The latter property is written as

$$\forall \mathbf{x}, \forall t = 1, \ldots, T, \quad f(\mathbf{x}) > 0 \implies g_t(\mathbf{x}) > 0$$

this property is referred to as the *conservation hypothesis* which means that all the g-classifiers should be implemented in order to preserve all the positive answers of the initial f-classifier.

We want $\{g_t\}_t$ to be both efficient and with equivalent error rates compared to f. Again, as the representational power of a g-network (w.r.t its associated f-network) is limited, seeking to obtain the same false positive (FP) and false

[3] In all this work, we use the Rectified Linear Unit (ReLU); the latter is defined as $h(\mathbf{x}) = \max(0, \mathbf{x})$.

[4] The representational power of a deep kernel network is closely related to the number of its parameters. As the number of parameters increases, the maximum number of samples that can be accurately shattered (whatever their labeling) becomes larger.

negative (FN) rates (w.r.t f) is clearly out of reach. Hence, we seek to make the FN rate of the g and f networks as close as possible (by implementing the conservation hypothesis) to the detriment of an increase of the FP rate of the g-network. With this setting, patterns classified as positive in the early stage of the cascade will further be processed through the subsequent stages and only those belonging (or resembling) to the targeted class will reach the final stage of the cascade; hence the overall evaluation cost of the cascade will be dominated by the cost to reject negative patterns using very cheap g-networks.

3.2. Learning cheap g-networks

Given a classifier f, we aim to design its surrogate cascade classifiers $\{g_t\}_t$ with fixed complexities (again defined by the depth and number of units; see Section 4). For a fixed stage t in the cascade, we rewrite its classifier g_t simply as g and the output of its g-network as κ_g. We propose to find the parameters of the g-network by minimizing the following loss

$$\min_{\mathbf{w}} \quad \beta^- \sum_{i=1}^{\ell} 1_{\{f(\mathbf{x}_i)\leq 0\}} \frac{1}{1+\exp(-\gamma g(\mathbf{x}_i))} + \beta^+ \sum_{i=1}^{\ell} 1_{\{f(\mathbf{x}_i)>0\}} \frac{\exp(-\gamma g(\mathbf{x}_i))}{1+\exp(-\gamma g(\mathbf{x}_i))} \tag{2}$$

here $\mathbf{w}$ are the weights of the g-network (as defined in Eq. 1), $f(\mathbf{x}) = \sum_i \alpha_i^f \mathbf{y}_i \kappa_f(\mathbf{x}, \mathbf{x}_i) + b_f$ is assumed pretrained and $g(\mathbf{x}) = \sum_i \alpha_i^g f(\mathbf{x}_i)\kappa_g(\mathbf{x}, \mathbf{x}_i) + b_g$ with $(\{\alpha_i^g\}_i, b_g)$ trained as shown in Section 3.3. In the above objective function, $\beta^+, \beta^- \geq 0$ and $\gamma \geq 0$ is the steepness of the logistic function $(1/(1+\exp(-\gamma g(\mathbf{x}))))$ set in practice to very large values, so this function acts as a smooth differentiable variant of the 0-1 loss. When $\beta^+ \gg \beta^-$, this re-balances the FP and FN rates in a way that favors very small FN of g w.r.t f (i.e., it makes it possible to implement the conservation hypothesis) to the detriment of an increase of the FP rate.

Note that this criterion is a variant of the cross-entropy loss; it seeks to reduce the number of contradictory outputs from the classifiers f and g. Note also that this criterion does not require labeled data provided that the classifier f (and its f-network) are already pretrained[5]; as shown in Section 3.3, the outputs of the classifier f are used as reference labels, so this optimization framework could benefit from very large unlabeled sets in order to make the estimate of the g-classifiers (and their g-networks) more accurate.

[5]Training the f-classifier and its f-network requires minor updates of the objective function 2 (and also 3); only the indicator function terms are set according to the actual labels of data in $\{\mathbf{x}\}$ (i.e., $1_{\{f(\mathbf{x})>0\}}$ is replaced by $1_{\{\mathbf{y}=+1\}}$ and $1_{\{f(\mathbf{x})\leq 0\}}$ by $1_{\{\mathbf{y}=-1\}}$).

3.3. Optimization

The goal is also to learn the SVM parameters $(\{\alpha_i^g\}_i, b_g)$ on top of the current estimate of κ_g. For that purpose, we use the forward information $\kappa_g(\cdot,\cdot)$ from the learned g-network in order to build a binary classifier g by minimizing a global hinge loss and a regularization term

$$\min_{g} \; C \sum_{i=1}^{\ell} \max\big(0, 1 - f(\mathbf{x}_i)g(\mathbf{x}_i)\big) + \frac{1}{2}\|g\|_{\mathcal{H}}^2, \tag{3}$$

here $C \geq 0$ controls the tradeoff between regularization and empirical error. According to the representer theorem [10], the dual form of Eq. 3 can be rewritten as

$$\max_{\alpha^g} \sum_{i=1}^{\ell} \alpha_i^g - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i^g \alpha_j^g f(\mathbf{x}_i) f(\mathbf{x}_j)\, \kappa_g(\mathbf{x}_i, \mathbf{x}_j)$$
$$\text{s.t.} \quad 0 \leq \alpha_i^g \leq C, \; \sum_{i=1}^{\ell} \alpha_i^g f(\mathbf{x}_i) = 0, \tag{4}$$

following the KKT conditions [4], b_g is a shift that guarantees the equilibrium constraint (in Eq. 4).

The objective functions 2 and 4 are optimized w.r.t two parameters: respectively weights $\mathbf{w}$ of the g-network and (α^g, b_g) of classifier g. Alternating optimization strategy is adopted, i.e., we fix $\mathbf{w}$ to optimize (α^g, b_g), and then vice-versa. At each iteration, when $\mathbf{w}$ is fixed, $\kappa_g(.,.)$ is also fixed, and (α^g, b_g) are optimized using an SVM solver (LIBSVM in practice [6]). When (α^g, b_g) are fixed, the gradient of Eq. 2 w.r.t the output k_g (of the g-network) is evaluated, and a round of backpropagation is achieved and $\mathbf{w}$ is accordingly updated using gradient descent. The iterative procedure (shown in algorithm 1) continues until convergence or when a maximum number of iterations is reached.

Algorithm 1: Deep Kernel Network Learning

Input: Initial $\mathbf{w}^{(l)}(l = 1, \ldots, L-1)$
Output: Optimal $\mathbf{w}^{(l)}(l = 1, \ldots, L-1)$, α^g, b_g

repeat
- Fix $\mathbf{w}$, compute the output kernel $\kappa_g(\mathbf{x}_i, \mathbf{x}_j), \forall i, j \in 1, \ldots \ell$;
- α^g, b_g are learned by the LIBSVM solver;
- Fix α^g, b_g, compute the gradient of 2 w.r.t $\{\kappa_g(\mathbf{x}_i, \mathbf{x}_j)\}_{ij}$;
- Update $\mathbf{w}$ (and hence κ_g) using backpropagation and gradient descent;

until *Convergence*;

Figure 3. This figure shows an example of a district hit by a tornado: (left) reference image, (middle) test image and (right) a mask of (a hand-labeled) change detection ground-truth shown in white (severe house damage).

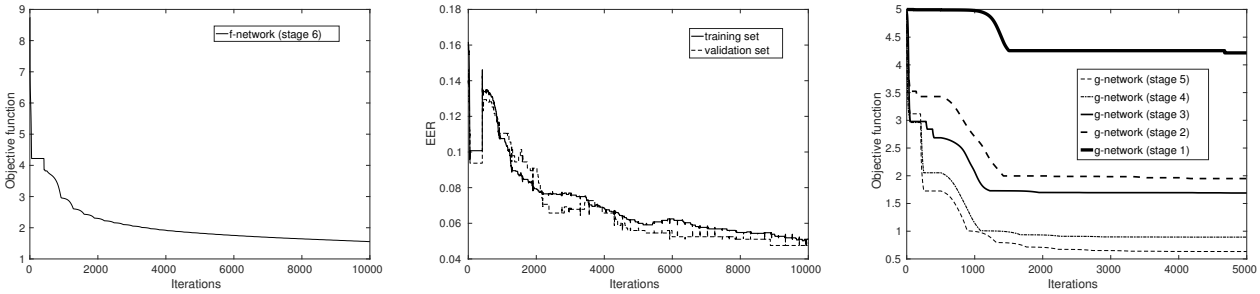

Figure 4. This figure shows (left) the evolution of the objective function 2 used to train the f-network w.r.t different iterations of optimization, (middle) the EER of the underlying f-classifier on training and validation sets, (right) the evolution of the objective functions 2 (used to train the g-networks) w.r.t different iterations of optimization.

4. Experiments

Datasets and Task: we evaluate the performance of our proposed method on the challenging task of satellite image change detection [35]. The goal is to find instances of relevant changes into a given scene acquired at instance t_1 with respect to the same scene taken at instant $t_0 < t_1$; these acquisitions (at instants t_0, t_1) are referred to as *reference* and *test* images respectively. This task is known to be very challenging due to the difficulty to characterize relevant changes (appearance or disappearance of objects[6]) from irrelevant ones (such as the presence of cars, clouds, etc.), and it is also very time demanding as the amount of data to process on large geographic areas is extremely large. Indeed, with the spread of remote sensors and unmanned aerial vehicles (UAV), and in the particular important scenario of damage assessment after natural hazards (such as tornadoes, earth quakes, etc.), it is crucial to achieve automatic change detection very promptly in order to organize and prioritize rescue operations; that's why one should use very accurate learning and classification algorithms (such as deep networks) while being able to process large amount of data efficiently.

Considering this scenario, we use a database $\mathcal{L} \cup \mathcal{U}$ of 680928 non-overlapping patch pairs (of 30×30 pixels in RGB) taken from six registered (reference and test) GeoEye-1 satellite images (of 9850×10400 pixels each). These images cover a very large area – of about 20×20 $\mathbf{km}^2$ – around Joplin (Missouri; see an example of a district from this area in Fig. 3) and show many changes after tornadoes that happened in may 2011 (building destruction, etc.) and no-changes (including irrelevant ones such as car appearance/disappearance, etc.). Each patch pair (in reference and test images) is encoded with 4096 coefficients corresponding to the difference between the outputs of the 4096-dimensional-layer (of the pretrained VGG-net [40]) on the reference and test patches. A given patch pair, denoted as $\mathbf{x}$ (with $\mathbf{x} \in \mathcal{U}$), is declared as a "change" or "no-change" depending on the scores of the trained SVM classifiers.

Evaluation measures: in order to evaluate the performances of our change detection classifiers, we use the following evaluation measures

- False alarm (FA) and detection rate (DR): the former is the fraction of ground-truth "no-changes" which are declared as positive while the latter is the fraction of ground-truth "changes" which are correctly classified as positive. Smaller FA and higher DR imply better

[6]This can be any object so there is no a priori knowledge about what object may appear or disappear into a given scene.

Figure 5. This figure shows the evolution of detections through the different stages of the cascade; as we go through different stages of the cascade the *global* number of false alarms decreases (in contrast to the setting of table 2, a classifier, at a given stage, is applied only to the patterns declared as positive by the preceding stages).

Figure 6. Figure, in the right-hand side, shows the amount of processing (number of stages used in the cascade) in order to reject or accept different patches in the test image (shown in the left-hand side); darker colors correspond to more intense processing.

performances.

- The equal error rate (EER) : this is defined as the average between FA and (1-DR). EER is the balanced generalization error that equally weights errors in "change" and "no-change" classes. Smaller EER implies better performance.
- The relative (rFA) and the conservation rate (cons): these two measures are similar to FA and DR respectively with the only difference being the ground-truth which is taken from the f-classifier instead of the original ground-truth.

all these measures are evaluated on the unlabeled data in $\mathcal{U}$.

Pretraining the f-network: this kernel network is fully connected and has 8 layers with 128 units per layer excepting the output layer which has a single unit; these layers consist of convolutional units followed by rectified linear units (ReLU). The 128 input kernels correspond to the values of the Gaussian similarity function[7] evaluated on

[7]with a scale factor set to the average distance between the VGG de-

128 disjoint chunks taken from the 4096 VGG dimensions; hence, each chunk has $4096/128 = 32$ dimensions.

In order to train the parameters of this f-network we consider a random subset $\mathcal{L}$ including 3000 patch pairs from the original set of 680928 patch pairs; $2/3$ of $\mathcal{L}$ are split into 10 mini-batches for training (i.e. to minimize the objective function 2) while the remaining $1/3$ is used as a validation set. The test error is reported on all the remaining $(680928-3000)$ patch pairs. The weights of the f-network, set initially as flat, are updated using stochastic gradient descent (SGD) and back-propagation as shown in algorithm (1); the latter is run iteratively for a maximum number of epochs (set in practice to 10000) in order to obtain convergence (see Fig. 4, left) and this is observed in less than one hour on a standard PC with a 3Ghz CPU. When training the f-network, β^+, β^- are set proportional to $\frac{1}{|\{\mathbf{y}_i=+1\}_i|}$, $\frac{1}{|\{\mathbf{y}_i=-1\}_i|}$ respectively and the step-size of SGD (denoted as ν) is set iteratively inversely proportional to the speed of change of the objective function 2; when this speed increases (resp. decreases), ν decreases as $\nu \leftarrow \nu \times 0.99$ (resp. increases as $\nu \leftarrow \nu/0.99$). Table 1 and Fig. (4, left/middle) show the evolution of the objective function as well as the equal error rates of the f-network w.r.t different iterations of the optimization process shown in algorithm (1).

	% EER (Training)	% EER (Validation)
Weights (initialization)	16.18	13.85
Weights (at convergence)	05.14	04.90

Table 1. This table shows the EER of the f-networks at initialization and at the end of the iterative process.

Training the cascade of the g-networks: in order to build the g-networks of our cascade, we consider the following architectures (complexities)

- **Stage 1:** this kernel network is fully connected and has 4 layers with 2 units per layer; these layers consist of convolutional units followed by rectified linear units (ReLU) excepting the output layer which has a single convolutional unit followed by a ReLU.
- **Stage 2:** this kernel is similar to the previous one except that the number of units per layer is 8.
- **Stage 3:** the only difference w.r.t stage 2 is the number of layers which is now set to 6.
- **Stage 4:** the only difference w.r.t stage 3 is the number of units per layer which is now set to 32.
- **Stage 5:** this kernel is similar to the previous one except that the number of units per layer is 64.

scriptors of data and their neighbors.

- **Stage 6:** this is exactly the pretrained f-network.

Similarly to the f-network, we use the same splits of data (into training, validation and test sets, etc.) in order to learn the parameters of the g-networks. These parameters, set initially as flat, are again updated using SGD and back propagation as shown in algorithm (1); in these experiments, the maximum number of epochs is now set to 5000 as the number of parameters in the g-networks is smaller compared to the f-network. With this setting, convergence is observed in less than 30mins using the same hardware configuration. In all these experiments, the step-size of SGD is set as described earlier while β^+, β^- are now set proportional to $\frac{0.99}{|\{f(\mathbf{x}_i)>0\}_i|}$, $\frac{0.01}{|\{f(\mathbf{x}_i)\leq 0\}_i|}$ in order to implement the conservation hypothesis.

Stage	%cons	%rFA	%DR	%FA	%ERR	time(ms)
1	99.16	41.04	97.55	43.56	23.00	745
2	98.86	37.37	96.88	40.09	21.61	897
3	98.77	35.27	96.70	38.07	20.68	1180
4	98.80	39.78	96.76	42.53	22.89	2721
5	98.70	28.24	96.56	31.46	17.45	4639
6	-	-	**97.14**	**04.44**	**03.65**	14230

Table 2. This table shows different evaluation measures of the g-networks w.r.t stages of the cascade as well as the average processing time. All these percentages are evaluated on the test set $\mathcal{U}$ while processing time is the average time to process a given pair of very large reference and test images (of 9850×10400 pixels); each reference and test image includes 113488 patches. Note that cons and rFA are not given for stage 6 as the g-network of this stage is exactly the f-network so these measures are obviously equal to 100% and 0% respectively. In order to study the behavior of different stages independently, each classifier is evaluated using all the data in $\mathcal{U}$, so the resulting FAs are not necessarily decreasing. In these experiments, the significant increase of FAs (from stage 6 to the other stages) is mainly due to the implementation of the conservation hypothesis that maintains a high detection rate to the detriment of an increase of false alarms.

Fig. (4, right) and Table (2) show respectively the evolution of the objective function (2) w.r.t different iterations of optimization and different evaluation measures of the underlying g-networks (obtained at convergence) w.r.t different stages of the cascade. We observe from these results that as we go deep in the cascade, the characteristics of the g-networks resemble more and more the original f-network; the efficiency decreases and the discrimination power (EER) remains stable or improves.
Fig. 5 shows examples of change detection results obtained through different stages of the cascade and Fig. 6 shows the amount of processing in order to classify different patches as changes or no-changes. From these results, it is clear that almost all the areas are rejected at the early stages of the cascade and only few areas (changes and change-like structures) require more intense processing. In practice, our coarse-to-fine cascade is almost $10\times$ faster than the original

	%DR	%FA	%EER	time (ms)
f-network+classifier	97.14	04.44	**03.65**	14230
cascade	92.16	03.80	**05.82**	1627 ($\sim 10\times$ faster)

Table 3. This table shows the overall performances of the f-network and the cascade of networks. Again, these percentages are evaluated on the test set $\mathcal{U}$ while processing time is the average time to process a given pair of very large reference and test images (of 9850×10400 pixels); each reference and test image includes 113488 patches.

f-network while its overall EER (shown in Table 3) remains relatively stable.

5. Conclusion

We introduced in this paper a novel approach for efficient deep kernel network evaluation. The design principle of our method is coarse-to-fine; it is based on a cascade of kernel networks and classifiers with increasing complexity and discrimination power. Networks in the early stages of the cascade are cheap and are used to reject many patterns efficiently while those belonging to the deep stages of the cascade are more expensive and more discriminating. The parameters of these networks are obtained by solving several cross-entropy minimization problems that reduce the difference between the original and the reduced cost kernel networks.

Even though tested on the particular (challenging) problem of change detection, this method is generic and could be extended to other imbalanced classification tasks such as object and rare event detection in still and video images where the untargeted classes are dominant. Other possible extensions of this work include transfer learning; indeed one may reduce the complexity of existing very deep networks (using our optimization framework) prior to achieve fine-tuning as this may reduce the computational cost of learning very significantly.

References

[1] Y. Amit and D. Geman. A computational model for visual selection. *Neural computation*, 11(7):1691–1715, 1999.

[2] Y. Amit, D. Geman, and X. Fan. A coarse-to-fine strategy for multiclass shape detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(12):1606–1621, 2004.

[3] F. R. Bach, G. R. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004.

[4] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[5] C. J. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. In *Advances in neural information processing systems*, pages 375–381, 1997.

[6] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011.

[7] S. Changpinyo, M. Sandler, and A. Zhmoginov. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*, 2017.

[8] Y. Cho and L. K. Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.

[9] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 539–546. IEEE, 2005.

[10] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[11] L. Deng, D. Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[12] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[13] F. Fleuret and D. Geman. Coarse-to-fine face detection. *International Journal of computer vision*, 41(1):85–107, 2001.

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

[15] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[16] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 243–254. IEEE Press, 2016.

[17] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[18] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in NIPS*, pages 1135–1143, 2015.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

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

[21] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE TPAMI*, 35(1):221–231, 2013.

[22] Y.-G. Jiang, Z. Wu, J. Wang, X. Xue, and S.-F. Chang. Exploiting feature and class relationships in video categorization with regularized deep neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[23] M. Jiu and H. Sahbi. Semi supervised deep kernel design for image annotation. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1156–1160. IEEE, 2015.

[24] M. Jiu and H. Sahbi. Laplacian deep kernel learning for image annotation. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 1551–1555. IEEE, 2016.

[25] M. Jiu and H. Sahbi. Nonlinear deep kernel learning for image annotation. *IEEE Transactions on Image Processing*, 26(4):1820–1832, 2017.

[26] C. Jose, P. Goyal, P. Aggrwal, and M. Varma. Local deep kernel learning for efficient non-linear svm prediction. In *International Conference on Machine Learning*, pages 486–494, 2013.

[27] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[28] W. Kienzle, M. O. Franz, B. Schölkopf, and G. H. Bakir. Face detection—efficient and rank deficient. In *Advances in NIPS*, pages 673–680, 2005.

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

[30] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. A convolutional neural network cascade for face detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[31] S. Z. Li and Z. Zhang. Floatboost learning and statistical face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1112–1123, 2004.

[32] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In *Advances in Neural Information Processing Systems*, pages 2627–2635, 2014.

[33] S. Romdhani, P. Torr, B. Scholkopf, and A. Blake. Computationally efficient face detection. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 695–700. IEEE, 2001.

[34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

[35] H. Sahbi. Misalignment resilient cca for interactive satellite image change detection. In *IEEE ICPR*, pages 3326–3331, 2016.

[36] H. Sahbi and N. Boujemaa. Coarse-to-fine support vector classifiers for face detection. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 3, pages 359–362. IEEE, 2002.

[37] H. Sahbi and D. Geman. A hierarchy of support vector machines for pattern detection. *Journal of Machine Learning Research*, 7(Oct):2087–2123, 2006.

[38] H. Sahbi, D. Geman, and N. Boujemaa. Face detection using coarse-to-fine support vector classifiers. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 3, pages 925–928. IEEE, 2002.

[39] S. Shi and X. Chu. Speeding up convolutional neural networks by exploiting the sparsity of rectifier units. *arXiv preprint arXiv:1704.07724*, 2017.

[40] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[41] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[42] E. V. Strobl and S. Visweswaran. Deep multiple kernel learning. In *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, volume 1, pages 414–417. IEEE, 2013.

[43] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[44] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[45] P. Viola and M. J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

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

[47] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016.

[48] C. Wu, W. Wen, T. Afzal, Y. Zhang, Y. Chen, and H. Li. A compact dnn: Approaching googlenet-level accuracy of classification and domain adaptation. *arXiv preprint arXiv:1703.04071*, 2017.

[49] N. Yu, S. Qiu, X. Hu, and J. Li. Accelerating convolutional neural networks by group-wise 2d-filter pruning. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 2502–2509. IEEE, 2017.

[50] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015.

[51] J. Zhuang, I. W. Tsang, and S. C. Hoi. Two-layer multiple kernel learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 909–917, 2011.