arXiv:1406.3332v1 [cs.CV] 12 Jun 2014

Convolutional Kernel Networks

Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid
Inria*
firstname.lastname@inria.fr

Abstract

An important goal in visual recognition is to devise image representations that are invariant to particular transformations. In this paper, we address this goal with a new type of convolutional neural network (CNN) whose invariance is encoded by a reproducing kernel. Unlike traditional approaches where neural networks are learned either to represent data or for solving a classification task, our network learns to approximate the kernel feature map on training data.

Such an approach enjoys several benefits over classical ones. First, by teaching CNNs to be invariant, we obtain simple network architectures that achieve a similar accuracy to more complex ones, while being easy to train and robust to overfitting. Second, we bridge a gap between the neural network literature and kernels, which are natural tools to model invariance. We evaluate our methodology on visual recognition tasks where CNNs have proven to perform well, *e.g.*, digit recognition with the MNIST dataset, and the more challenging CIFAR-10 and STL-10 datasets, where our accuracy is competitive with the state of the art.

1 Introduction

We have recently seen a revival of attention given to convolutional neural networks (CNNs) [22], due to their success in large-scale visual recognition tasks [15, 21, 29]. The architecture of CNNs is relatively simple and consists of successive layers organized in a hierarchical fashion. Each layer involves convolutions with learned filters followed by a non-linearity, and downsampling operations called "feature pooling". The resulting image representation seems to achieve some invariance to image perturbations and to encode complex visual patterns [32]. Training CNNs remains however difficult since high-capacity networks may involve billions of parameters to learn, which requires both high computational power, *e.g.*, GPUs, and appropriate regularization techniques [18, 21, 29].

The exact nature of invariance achieved by CNNs is also not well understood. Only recently, the invariance of some related architectures has been characterized; this is the case for the wavelet scattering transform [8], or the hierarchical models of [7]. Our work revisits convolutional neural networks, but adopts a significantly different approach than the traditional one. We use indeed kernels [26], which are natural tools to model invariance [14]. Inspired by the hierarchical kernel descriptors of [2], we propose a reproducing kernel that produces invariant multi-layer image representations.

Our main contribution is an approximation scheme called *convolutional kernel network* (CKN) to make the kernel approach computationally feasible. Such a scheme turns out to be a new type of CNN, which differs from classical ones in its objective function. The network is trained to linearly approximate the kernel, and thus the procedure involves no supervision. Another difference is in the non-linear functions that we use. Interestingly, they resemble rectified linear units [1, 29], even though they naturally emerge from the approximation scheme and were not handcrafted.

As a result, we bridge a gap between kernel methods and neural networks, and we believe that such a direction is fruitful for the future. Our network is learned without supervision since the label

*LEAR team, Inria Grenoble, Laboratoire Jean Kuntzmann, CNRS, Univ. Grenoble Alpes, France.

information is only used subsequently in a support vector machine (SVM). Yet, we achieve near state-of-the-art results on several datasets such as MNIST [22], CIFAR-10 [20] and STL-10 [13] with simple architectures, few parameters to learn, and no data augmentation. Open-source code for learning our convolutional kernel networks will be provided upon publication of the paper.

1.1 Related Work

There were several attempts in the past to build kernel-based methods that mimic deep neural networks; we only review here the most related to our approach.

Arc-cosine kernels. Kernels for building deep large-margin classifiers have been introduced in [10]. For any pair of vectors, the single-layer arc-cosine kernel relies on an integral representation, and multilayer extensions are built by multiple kernel compositions. Similarly, our kernels rely on a integral representation, and enjoy a multilayer construction. However, in contrast to arc-cosine kernels: (i) we build our sequence of kernels by *convolutions*, using local information over spatial neighborhoods (as opposed to compositions, using global information); (ii) we propose a new training procedure for learning a compact representation of the kernel in a *data-dependent* manner.

Multilayer derived kernels. Kernels that enjoy invariance properties for visual recognition were proposed in [7]. Such kernels are built with a parameterized "neural response" function, which consists in computing the maximal response of a base kernel over a local neighborhood. Multiple layers are then built by iteratively renormalizing the response kernels and pooling using neural response functions. Learning is performed by plugging the obtained kernel in a SVM. In contrast to [7], we propagate information up, from lower to upper layers, by using sequences of convolutions. Furthermore, we propose a simple and effective data-dependent way to learn a compact representation of our kernels and show that we obtain near state-of-the-art performance on several benchmarks.

Hierarchical kernel descriptors. The kernels proposed in [2, 3] produce multilayer image representations for visual recognition tasks. We discuss in details these kernels in the next section: our paper generalizes them and establishes a strong link with convolutional neural networks.

2 Convolutional Multilayer Kernels

The convolutional multilayer kernel is a generalization of the hierarchical kernel descriptors introduced in computer vision [2, 3]. The kernel produces a sequence of image representations that are built one on top of each other in a multilayer fashion. Each layer can be interpreted as a non-linear transformation of the previous one with additional invariance. We call these layers *image feature maps*[1], and formally define them as follows:

Definition 1. *An image feature map φ is a function $\varphi : \Omega \to \mathcal{H}$, where Ω is a (usually discrete) subset of $[0,1]^d$ representing "coordinates" in the image and $\mathcal{H}$ is a Hilbert space.*

For all practical examples in this paper, Ω is a two-dimensional grid and corresponds to different locations in a two-dimensional image. In other words, Ω is a set of pixel coordinates. Given $\mathbf{z}$ in Ω, the point $\varphi(\mathbf{z})$ corresponds to some characteristics of the image at location $\mathbf{z}$, or in a neighborhood of $\mathbf{z}$. For instance, a color image of size $m \times n$ with three channels, red, green, and blue, may be represented by an initial feature map $\varphi_0 : \Omega_0 \to \mathcal{H}_0$, where Ω_0 is an $m \times n$ regular grid, $\mathcal{H}_0$ is the Euclidean space $\mathbb{R}^3$, and φ_0 provides the color pixel values. With the multilayer scheme, non-trivial feature maps will be obtained subsequently, which will encode more complex image characteristics. With this terminology in hand, we now introduce the convolutional kernel, first, for a single layer.

Definition 2 (Convolutional Kernel with Single Layer). *Let us consider two images represented by two image feature maps, respectively φ and $\varphi' : \Omega \to \mathcal{H}$, where Ω is a set of pixel locations, and $\mathcal{H}$ is a Hilbert space. The one-layer convolutional kernel between φ and φ' is defined as*

$$K(\varphi, \varphi') := \sum_{\mathbf{z} \in \Omega} \sum_{\mathbf{z}' \in \Omega} \|\varphi(\mathbf{z})\|_{\mathcal{H}} \|\varphi'(\mathbf{z}')\|_{\mathcal{H}} \, e^{-\frac{1}{2\beta^2}\|\mathbf{z}-\mathbf{z}'\|_2^2} e^{-\frac{1}{2\sigma^2}\|\tilde{\varphi}(\mathbf{z})-\tilde{\varphi}'(\mathbf{z}')\|_{\mathcal{H}}^2}, \tag{1}$$

[1] In the kernel literature, "feature map" denotes the mapping between data points and their representation in a RKHS [26]. Here, feature maps refer to spatial maps, as usual in the neural network literature [22].

where β and σ are smoothing parameters of Gaussian kernels, and $\tilde{\varphi}(\mathbf{z})$ and $\tilde{\varphi}(\mathbf{z}')$ are normalized versions of $\varphi(\mathbf{z})$ and $\varphi'(\mathbf{z}')$, respectively. More precisely, $\tilde{\varphi}(\mathbf{z}) := (1/\max(\|\varphi(\mathbf{z})\|_{\mathcal{H}}, \varepsilon))\, \varphi(\mathbf{z})$, and the definition is similar for $\varphi'(\mathbf{z}')$.[2]

It is easy to see that K is a positive definite kernel since it only involves sums and products of positive definite kernels [26]. It compares normalized features $\tilde{\varphi}(\mathbf{z})$ and $\tilde{\varphi}'(\mathbf{z}')$ extracted at all locations $\mathbf{z}$ and $\mathbf{z}'$ with a Gaussian kernel. Another Gaussian function compares the locations $\mathbf{z}$ and $\mathbf{z}'$ providing invariance to local deformations. Indeed, when β goes to infinity, the kernel becomes invariant to the positions $\mathbf{z}$ and $\mathbf{z}'$ of the features $\varphi(\mathbf{z})$ and $\varphi'(\mathbf{z}')$. When β is small, only features placed at the same location $\mathbf{z} = \mathbf{z}'$ are compared to each other, and the kernel has no shift-invariance. Before moving to a model with additional layers, let us present a few concrete examples when applying the convolutional kernel to simple input feature maps $\varphi_0 : \Omega_0 \to \mathcal{H}_0$.

Gradient map. Assume that $\mathcal{H}_0 = \mathbb{R}^2$ and that $\varphi_0(\mathbf{z})$ provides the two-dimensional gradient of the image at pixel $\mathbf{z}$, which is often computed with first-order differences along each dimension. Then, the quantity $\|\varphi_0(\mathbf{z})\|_{\mathcal{H}_0}$ is the gradient intensity, and $\tilde{\varphi}_0(\mathbf{z})$ is its orientation, which can be characterized by a particular angle—that is, there exists θ in $[0; 2\pi]$ such that $\tilde{\varphi}_0(\mathbf{z}) = [\cos(\theta), \sin(\theta)]$. The resulting kernel K is exactly the kernel descriptor introduced in [2, 3] for natural image patches.

Patch map. In that setting, φ_0 associates to a location $\mathbf{z}$ an image patch of size $m \times m$ centered at $\mathbf{z}$. Then, the space $\mathcal{H}_0$ is simply $\mathbb{R}^{m \times m}$, and $\tilde{\varphi}_0(\mathbf{z})$ is a *contrast-normalized* version of the patch, which is a useful transformation for visual recognition according to classical findings in computer vision [19]. When the image is encoded with three color channels, patches are of size $m \times m \times 3$.

We now define the multilayer convolutional kernel, generalizing some ideas of [2].

Definition 3 (**Multilayer Convolutional Kernel**). *Let us consider a set $\Omega_{k-1} \subseteq \mathbb{R}^d$ and a Hilbert space $\mathcal{H}_{k-1}$. We build a new set Ω_k and a new Hilbert space $\mathcal{H}_k$ as follows:*

(i) choose a patch shape $\mathcal{P}_k$ defined as a bounded symmetric subset of $[0,1]^d$, and a set of coordinates Ω_k such that for all $\mathbf{z}_k$ in Ω_k, the patch $\{\mathbf{z}_k\} + \mathcal{P}_k$ is a subset of Ω_{k-1};[3]

(ii) define the convolutional kernel K_k on the "patch" feature maps $\mathcal{P}_k \to \mathcal{H}_{k-1}$, by replacing in (1), Ω by $\mathcal{P}_k$, $\mathcal{H}$ by $\mathcal{H}_{k-1}$, and σ, β by appropriate smoothing parameters σ_k, β_k. We denote by $\mathcal{H}_k$ the Hilbert space for which K_k is a reproducing kernel.

An image represented by a feature map $\varphi_{k-1} : \Omega_{k-1} \to \mathcal{H}_{k-1}$ at layer $k-1$ is now encoded in the k-th layer as $\varphi_k : \Omega_k \to \mathcal{H}_k$, where for all $\mathbf{z}_k$ in Ω_k, $\varphi_k(\mathbf{z}_k)$ is the representation in $\mathcal{H}_k$ of the patch feature map $\mathbf{z} \mapsto \varphi_{k-1}(\mathbf{z}_k + \mathbf{z})$ for $\mathbf{z}$ in $\mathcal{P}_k$.

Concretely, the kernel K_k between two patches of φ_k and φ'_k at respective locations $\mathbf{z}_k$ and $\mathbf{z}'_k$ is

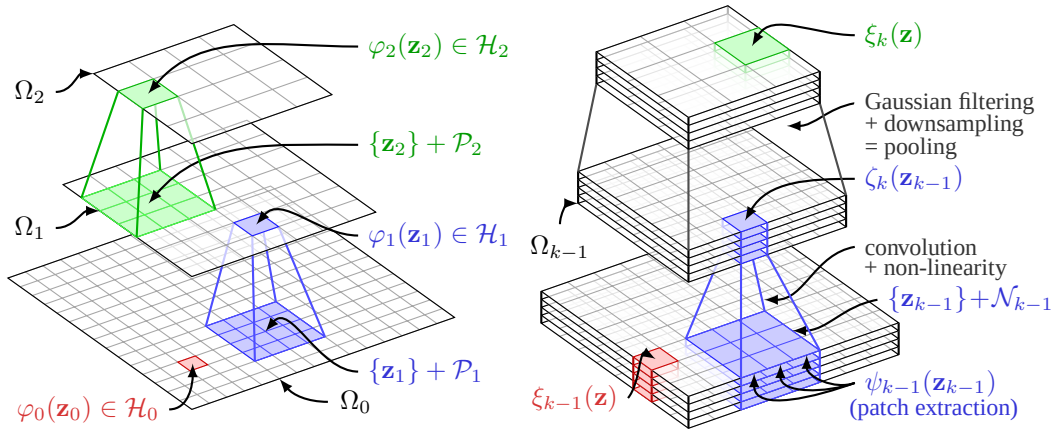
$$\sum_{\mathbf{z} \in \mathcal{P}_k} \sum_{\mathbf{z}' \in \mathcal{P}_k} \|\varphi_k(\mathbf{z}_k + \mathbf{z})\|_{\mathcal{H}_k} \|\varphi'_k(\mathbf{z}'_k + \mathbf{z}')\|_{\mathcal{H}_k} e^{-\frac{1}{2\beta_k^2}\|\mathbf{z}-\mathbf{z}'\|_2^2} e^{-\frac{1}{2\sigma_k^2}\|\tilde{\varphi}_k(\mathbf{z}_k+\mathbf{z}) - \tilde{\varphi}'_k(\mathbf{z}'_k+\mathbf{z}')\|_{\mathcal{H}_k}^2}. \quad (2)$$

In Figure 1(a), we illustrate the interactions between the different sets of coordinates Ω_k, patches $\mathcal{P}_k$, and feature spaces $\mathcal{H}_k$ across layers. For two-dimensional grids, a typical patch shape is a square, for example $\mathcal{P} := \{-1/n, 0, 1/n\} \times \{-1/n, 0, 1/n\}$ for a 3×3 patch in an image of size $n \times n$. Information encoded in the k-th layer differs from the $(k-1)$-th one in two aspects: first, each point $\varphi_k(\mathbf{z}_k)$ in layer k contains information about several points from the $(k-1)$-th layer and can possibly represent more complex patterns; second, the new feature map is more shift-invariant than the previous one due to the term involving the parameter β_k in Eq. (2).

The multilayer convolutional kernel slightly differs from the hierarchical kernel descriptors of [2] but exploits similar ideas. Bo et al. [2] define indeed several ad hoc kernels for representing local information in images, such as gradient, color, or shape. These kernels are close to the one defined in (1) but with a few variations. Some of them do not use normalized features $\tilde{\varphi}(\mathbf{z})$, and these kernels use different weighting strategies for the summands of (1) that are specialized to the image modality, *e.g.*, color, or gradient, whereas we use the same weight $\|\varphi(\mathbf{z})\|_{\mathcal{H}} \|\varphi'(\mathbf{z}')\|_{\mathcal{H}}$ for all kernels. We propose instead the generic formulation (1), which appears to perform well in practice. We believe that such a generalization is useful per se, but our main contribution comes in the next section, where we use the kernel as a new tool for learning invariant convolutional neural networks.

[2] When Ω is not discrete, the notation $\sum$ in (1) should be replaced by the Lebesgue integral $\int$ in the paper.

[3] For two sets A and B, the Minkowski sum $A + B$ is defined as $\{a + b : a \in A, b \in B\}$.



(a) Hierarchy of Image Feature Maps

(b) Zoom between layer k and $k-1$ of our CKN

Figure 1: Left: concrete representation of the successive layers for the multilayer convolutional kernel. Right: one layer of the convolutional neural network that approximates the kernel.

3 Training Invariant Convolutional Kernel Networks

We now show that a natural approximation scheme for the multilayer convolutional kernel gives rise to a convolutional neural network. In other words, the approximation can be achieved with a sequence of spatial convolutions with learned filters, non-linearities, and pooling operations.

Several schemes have been proposed for approximating a non-linear kernel with a linear one, such as the Nyström method and its variants [5, 30], which consist of projecting the data onto a finite-dimensional subspace of the Hilbert space for which the kernel is reproducing. Random sampling techniques in the Fourier domain for shift-invariant kernels are also popular [24]. Being able to linearly approximate convolutional kernels is critical because computing the full kernel matrix on a database of images is computationally infeasible, even for a moderate number of images ($\approx 10\,000$) and moderate number of layers. For this reason, Bo et al. [2] use the Nyström method for their hierarchical kernel descriptors. In our paper, we show that a particular convolutional neural network fits well the structure of the convolutional kernel, leading to an approach enjoying classical benefits of CNNs such as efficient prediction at test time.

3.1 Fast Approximation of the Gaussian Kernel

A recurrent component of our formulation is the Gaussian kernel. In this section, we show that an approximation scheme involves a linear operation with learned filters followed by a pointwise non-linearity. Our starting point is the next lemma, which can be obtained after a simple calculation.

Lemma 1 (**Expansion of the Gaussian Kernel**). *For all* $\mathbf{x}$ *and* $\mathbf{x}'$ *in* $\mathbb{R}^m$, *and* $\sigma > 0$,

$$e^{-\frac{1}{2\sigma^2}\|\mathbf{x}-\mathbf{x}'\|_2^2} = \left(\frac{2}{\pi\sigma^2}\right)^{\frac{m}{2}} \int_{\mathbf{w}\in\mathbb{R}^m} e^{-\frac{1}{\sigma^2}\|\mathbf{x}-\mathbf{w}\|_2^2} e^{-\frac{1}{\sigma^2}\|\mathbf{x}'-\mathbf{w}\|_2^2} d\mathbf{w}. \tag{3}$$

The lemma gives us an infinite-dimensional linear representation $[\sqrt{C}e^{-(1/\sigma^2)\|\mathbf{x}-\mathbf{w}\|_2^2}]_{\mathbf{w}\in\mathbb{R}^m}$ for all $\mathbf{x}$ in $\mathbb{R}^m$, where C is the constant in front of the integral. To obtain a finite-dimensional representation, we need to approximate the integral with a weighted finite sum, which is a classical problem in numerical analysis and statistics (see [28] and chapter 8 of [6] for a review). We choose instead a data-driven approach, and consider two different scenarios:

Small dimension, $m \leq 2$**.** When the data lives in a compact set of $\mathbb{R}^m$, the integral in (3) can be approximated by uniform sampling over a large enough set. We choose such a strategy for two types of kernels from Eq. (1): (i) the spatial kernels $e^{-(1/2\beta^2)\|\mathbf{z}-\mathbf{z}'\|_2^2}$; (ii) the term $e^{-(1/2\sigma^2)\|\tilde{\varphi}(\mathbf{z})-\tilde{\varphi}(\mathbf{z})'\|_{\mathcal{H}}^2}$ when φ is the "gradient map" presented in Section 2. In the latter case, $\mathcal{H} = \mathbb{R}^2$ and $\tilde{\varphi}(\mathbf{z})$ is the gradient orientation. We typically sample a few orientations as explained in Section 3.4.

Higher dimensions. When the dimension is high, uniform sampling suffers from the curse of dimensionality. Therefore, we choose to leverage the intrinsic low-dimensionality of the data. We learn importance weights $\boldsymbol{\eta} = [\eta_l]_{l=1}^p$ in $\mathbb{R}_+^p$ and sampling points $\mathbf{W} = [\mathbf{w}_l]_{l=1}^m$ in $\mathbb{R}^{m\times p}$ on training data $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in $\mathbb{R}^m$ with the following data-driven non-convex formulation

$$\min_{\boldsymbol{\eta}\in\mathbb{R}_+^p, \mathbf{W}\in\mathbb{R}^{m\times p}} \left[\frac{1}{n^2} \sum_{(i,j)} \left(e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i-\mathbf{x}_j\|_2^2} - \sum_{l=1}^p \eta_l e^{-\frac{1}{\sigma^2}\|\mathbf{x}_i-\mathbf{w}_l\|_2^2} e^{-\frac{1}{\sigma^2}\|\mathbf{x}_j-\mathbf{w}_l\|_2^2} \right)^2 \right]. \quad (4)$$

The resulting representation of a new data point $\mathbf{x}$ is simply the vector $[\sqrt{\eta_l} e^{-(1/\sigma^2)\|\mathbf{x}-\mathbf{w}_l\|_2^2}]_{l=1}^p$ in $\mathbb{R}^p$. We use the method (4) for approximating the Gaussian kernels $e^{-(1/2\sigma^2)\|\tilde{\varphi}(\mathbf{z})-\tilde{\varphi}'(\mathbf{z}')\|_{\mathcal{H}}^2}$ in (1), by proceeding as follows: first, we assume that we already have a finite-dimensional approximation of $\tilde{\varphi}(\mathbf{z})$—say, of dimension m—which we denote by $\tilde{\psi}(\mathbf{z})$; second, we apply (4) to the kernel $e^{-(1/2\sigma^2)\|\tilde{\psi}(\mathbf{z})-\tilde{\psi}'(\mathbf{z}')\|_2^2}$ by using a database of features $\tilde{\psi}(\mathbf{z})$ in $\mathbb{R}^m$ obtained from training data.

We can now start relating our approach to neural networks. After learning the parameters $\mathbf{W}$ and $\boldsymbol{\eta}$, computing the finite-dimensional approximation in $\mathcal{H}$ of a unit-norm vector $\mathbf{x}$ only involves a linear operation followed by a non-linearity. Indeed, the quantities $e^{-(1/\sigma^2)\|\mathbf{x}-\mathbf{w}_l\|_2^2}$ can be written as $f_l(\mathbf{w}_l^\top \mathbf{x})$, where f_l is the real-valued function $u \mapsto e^{-(1+\|\mathbf{w}_l\|_2^2)/\sigma^2+2u/\sigma^2}$. Since we apply (4) to normalized data $\tilde{\psi}(\mathbf{z})$, we expect the ℓ_2-norm of the sampling points $\mathbf{w}_l$ to to be close to one, and the functions f_l to have the form: $u \mapsto e^{(2/\sigma^2)(u-1)}$ for $u = \mathbf{w}_l^\top \mathbf{x}$ in $[-1, 1]$. In Figure 2, we show that we obtain a shape resembling the "rectified linear unit" function used in neural networks [29].

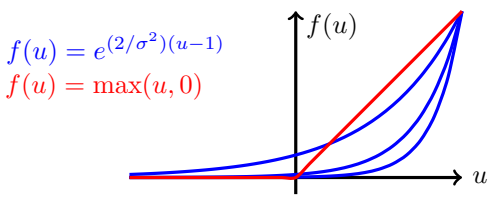


Figure 2: In red, we plot the function $u \mapsto \max(u, 0)$ often called "rectified linear unit" [29]. In blue, we plot non-linear functions used by our network. All plots are for u in $[-1, 1]$.

3.2 Approximating the Single-Layer Convolutional Kernel

With the methodology presented in the previous section, we now introduce an approximation scheme for the convolutional kernel $K(\varphi, \varphi')$ from Definition 2. We proceed as follows:

(i) we assume that for all $\varphi(\mathbf{z})$ and $\varphi'(\mathbf{z}')$, we already know some finite-dimensional approximations $\psi(\mathbf{z})$ and $\psi'(\mathbf{z}')$ in $\mathbb{R}^m$;

(ii) the kernels $e^{-(1/2\beta^2)\|\mathbf{z}-\mathbf{z}'\|_2^2}$ are approximated by uniform sampling: we define a set of equally spaced points $\mathcal{N}$, such that $e^{-(1/2\beta^2)\|\mathbf{z}-\mathbf{z}'\|_2^2} \approx C' \sum_{\mathbf{c}\in\mathcal{N}} e^{-(1/\beta^2)\|\mathbf{z}-\mathbf{c}\|_2^2} e^{-(1/\beta^2)\|\mathbf{z}'-\mathbf{c}\|_2^2}$, and C' is a constant independent of $\mathbf{z}$ and $\mathbf{z}'$.

(iii) we learn some weights $\boldsymbol{\eta}$ in $\mathbb{R}_+^p$ and sampling points $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_p]$ in $\mathbb{R}^{m\times p}$ to approximate the kernels $e^{-(1/2\sigma^2)\|\tilde{\varphi}(\mathbf{z})-\tilde{\varphi}'(\mathbf{z}')\|_{\mathcal{H}}^2}$ following the approach of the previous section.

Note that all parameter choices, such as $\mathcal{N}$, β, and σ, are discussed in Section 3.4. By plugging the above approximations in Eq. (1) and interchanging the sums, we get

$$K(\varphi, \varphi') \approx C' \sum_{\mathbf{c}\in\mathcal{N}} \sum_{l=1}^p \left[\sum_{\mathbf{z}\in\Omega} \zeta^l(\mathbf{z}) e^{-(1/\beta^2)\|\mathbf{c}-\mathbf{z}\|_2^2} \right] \left[\sum_{\mathbf{z}'\in\Omega} \zeta^l(\mathbf{z}') e^{-(1/\beta^2)\|\mathbf{c}-\mathbf{z}'\|_2^2} \right], \quad (5)$$

where we have introduced the quantity

$$\zeta^l(\mathbf{z}) := \|\psi(\mathbf{z})\|_2 \sqrt{\eta_l} e^{-(1/\sigma^2)\|\tilde{\psi}(\mathbf{z})-\mathbf{w}_l\|_2^2},$$

and also, by analogy, the quantity $\zeta^{l\prime}(\mathbf{z}')$ for φ'. Each function $\zeta^l : \Omega \to \mathbb{R}$ can be interpreted as a spatial map, where $\zeta^l(\mathbf{z})$ represents a non-linear filter response involving the non-linear functions

presented in Figure 2. According to (5), we need to subsample these maps ζ^l at some points $\mathcal{N}$ after convolving them with the two-dimensional filter $\mathbf{z} \mapsto \sqrt{C'}e^{-(1/\beta^2)\|\mathbf{z}\|_2^2}$, leading to a new set of p maps denoted by $\xi^l : \mathcal{N} \to \mathbb{R}$. In the same way, a set of p maps $\xi^{l\prime}$ are obtained for φ', and

$$K(\varphi, \varphi') \approx \sum_{\mathbf{c}\in\mathcal{N}} \sum_{l=1}^{p} \xi^l(\mathbf{c})\xi^{l\prime}(\mathbf{c}'). \tag{6}$$

The maps ζ^l and ξ^l coincide with the terminology of "feature maps" from neural networks. Building the maps ξ^l from ζ^l is achieved by Gaussian filtering and subsampling at predefined points $\mathbf{c}$ in $\mathcal{N}$, which is also called a "linear pooling step" in neural networks, and "downsampling" with a Gaussian anti-aliasing filter in signal processing. In the next section, we show that approximating the multilayer convolutional kernel can be achieved similarly, and leads to a particular CNN.

3.3 Convolutional Kernel Networks

We have seen that for approximating the single-layer convolutional kernel on a feature map $\varphi_0 : \Omega_0 \to \mathcal{H}_0$ where $\mathcal{H}_0$ is the Euclidean space $\mathbb{R}^{m_0}$, we need to (i) compute the linear responses $\tilde{\varphi}_0(\mathbf{z})^\top \mathbf{w}_l$; (ii) apply a pointwise non-linearity to obtain the maps ζ^l; (iii) perform linear pooling with Gaussian filtering to obtain new maps ξ^l. When in addition φ_0 is the "patch map" defined in Section 2, the vectors $\mathbf{w}_l$ can be interpreted as spatial filters, and the dot products $\tilde{\varphi}_0(\mathbf{z})^\top \mathbf{w}_l$ as convolutions. As a result, we have just described a one-layer convolutional neural network.

When the kernel involves several layers $\varphi_0, \ldots, \varphi_k$, as in Definition 3, the methodology of the previous section allows us to build a linear approximation of the kernel $K(\varphi_k, \varphi_k')$ under one condition: we need to know in advance finite-dimensional approximations $\psi_k(\mathbf{z}_k)$ of $\varphi_k(\mathbf{z}_k)$ for all $\mathbf{z}_k$ in Ω_k. We assume that such a condition is always satisfied for $\mathcal{H}_0$, which we choose finite-dimensional. Therefore, it remains to show how to build $\psi_k(\mathbf{z}_k)$ when the condition holds for $k-1$. From Definition 3, $\varphi_k(\mathbf{z}_k)$ in $\mathcal{H}_k$ is the representation derived from the kernel K_k presented in Eq. (2) of the patch feature map $\mathbf{z} \mapsto \varphi_{k-1}(\mathbf{z}_k + \mathbf{z})$. To achieve our goal, we apply the methodology of the previous section to K_k, by replacing in (5) Ω by $\mathcal{P}_k$, β by β_k, σ by σ_k, p by p_k, and $\mathcal{N}$ by $\mathcal{N}_k$. Then, we obtain an approximation $\psi_k(\mathbf{z}_k)$ of dimension $m_k = |\mathcal{N}_k|p_k$.

Note that since the patches may overlap, computing the vectors $\psi_k(\mathbf{z}_k)$ can be done efficiently for a given image represented by $\varphi_{k-1} : \Omega_{k-1} \to \mathcal{H}_{k-1}$. As in the previous section, let us define the maps $\zeta_k^l : \Omega_{k-1} \to \mathbb{R}$ as $\zeta_k^l(\mathbf{z}) := \|\psi_{k-1}(\mathbf{z})\|_2\sqrt{\eta_l}e^{-(1/\sigma_k^2)\|\tilde{\psi}_{k-1}(\mathbf{z})-\mathbf{w}_l\|_2^2}$ for all $\mathbf{z}$ in Ω_{k-1} and $l = 1, \ldots, p_k$. Given the maps ζ_k^l, all vectors $\psi_k(\mathbf{z}_k)$ can be computed in $O(|\Omega_k||\mathcal{N}_k||\mathcal{P}_k|p_k)$ operations, whereas the maps ζ_k^l require storing $O(|\Omega_{k-1}|p_k)$ real values.

At this point, we have obtained a computationally feasible approximation scheme that is fairly general regarding the choice of domains Ω_k and patches $\mathcal{P}_k$. However, a last approximation allows us to fill in the gap between our approach and multilayer CNNs under a few conditions: (i) Ω_{k-1} and Ω_k are grids; (ii) $\mathcal{P}_k$ is a square with same spacing as Ω_{k-1}; (iii) $\mathcal{N}_k$ is a square containing 0 with same spacing as Ω_k; Then, we can build the maps ξ_l^k as in the previous section, by convolving the maps ζ_k^l with the Gaussian filter $\mathbf{z} \mapsto \sqrt{C'}e^{-(1/\beta^2)\|\mathbf{z}\|_2^2}$, followed by subsampling with the same spacing as Ω_k. Such a construction is illustrated in Figure 1(b). As a result, a $|\mathcal{N}_k|p_k$-dimensional approximation of $\varphi_k(\mathbf{z}_k)$ is simply the $p_k \times \mathcal{N}_k$ patch $[\xi_k^l(\mathbf{z}_k + \mathbf{z})]_{l,\mathbf{z}\in\mathcal{N}_k}$. Consequently, obtaining the desired approximation $\psi_k(\mathbf{z}_k)$ requires extracting a patch from p_k maps of size $O(|\Omega_k|)$, and the dot products $\mathbf{w}_l^\top \psi_k(\mathbf{z}_k)$ are spatial convolutions on the "pooled" maps ξ_k^l. We have now obtained the convolutional kernel network displayed in Figure 1(b).

3.4 Practical Implementation: Parameter Setting and Optimization

The parameters that our convolutional kernel network requires are the following quantities: N (number of layers), β_k, σ_k (smoothing parameters); p_k (number of filters per layer), $\mathcal{N}_k$ (to obtain $\psi_k(\mathbf{z}_k)$ from the maps ξ_k^l), the downsampling factor between ξ_k^l and ζ_k^l, and the initial maps ζ_0^l. Other parameters appearing in the original kernel formulation of Section 2, such as $\mathcal{P}_k$, Ω_k, do not need to be chosen since they are implicitly linked with the ones we have just listed.

First, N, p_k, $\mathcal{N}_k$, and the downsampling factors are classical parameters of CNNs, which are left to the discretion of the user. The sets $\mathcal{N}_k$ represent patches on the maps ξ_k^l, as shown in Figure 1(b),

and are typically small. We tried the sizes $m \times m$ with $m = 3, 4, 5$ for the first layer, and $m = 2, 3$ for the upper ones. The number of filters p_k in our experiments is in the set $\{50, 100, 200, 400, 800\}$. The downsampling factor is always chosen to be 2 between two consecutive layers, whereas the last layer is downsampled to produce final maps ξ_N^l of a small size—say, 5×5 or 4×4.

The initial map ζ_0^l also needs to be set up. When using the "patch map" of Section 2, the maps ζ_0^l are simply the input image and $\mathcal{N}_1$ defines the patch size. For the "gradient map", we have $p_0 = 2$, the maps ζ_0^l carry the image gradient, and we set $\mathcal{N}_1 = \{0\}$, representing a 1×1 patch.

Finally, the parameters β_k are chosen according to signal processing principles. The purpose of the filter $\mathbf{z} \rightarrow\mapsto e^{-(1/\beta^2)\|\mathbf{z}\|_2^2}$ is indeed to remove high frequencies from the maps ζ_k^l before subsampling. When downsampling by a factor κ, we simply choose β to be κ times the spacing of two pixels on ζ_k^l. Similarly, the parameter σ_1 when sampling p_1 orientations for the "patch maps" is set to $2\pi/p_1$, where $p_1 = 12$ in all our experiments. The parameter σ_k in (4) is chosen according to a heuristic rule. We set σ_k to the 0.1 quantile of the training data $\|\psi_{k-1}(\mathbf{z}) - \psi'_{k-1}(\mathbf{z}')\|_2$.

Regarding the optimization problem (4), we learn the parameters $\mathbf{W}, \boldsymbol{\eta}$ of the layers from bottom to top. Stochastic gradient descent (SGD) is a natural candidate for this task since an enormous amount of training data is available. For the purpose of this paper, we have preferred to use L-BFGS-B [9] on $300\,000$ pairs of randomly selected training data points, and initializing the weights $\mathbf{W}$ with the K-means algorithm. L-BFGS-B is a parameter-free state-of-the-art batch method, which is probably not as fast as SGD, but which is much easier to use. Our goal was to demonstrate the performance of a new type of convolutional network, and we leave as future work any speed improvement.

4 Experiments

In this section, we present experiments on visual recognition tasks, and start by visualizing the filters we learn on natural image patches. All experiments were performed using a Matlab implementation, which will be made available upon publication of the paper, and an L-BFGS-B solver [9], interfaced in Matlab by Stephen Becker. Since our work is not focused on speed, we run the L-BFGS-B algorithm for $4\,000$ iterations, which seems to ensure convergence of the objective function to a stationary point. The image representations obtained with our convolutional kernel network, represented by the last map ξ^k, are used in a linear support vector machine. We use the software package LibLinear [16] modified to handle dense matrices. The image representations are centered, rescaled to have unit ℓ_2-norm on average, and the parameter C of the SVM is always selected on a validation set or by 5-fold cross-validation in the range 2^i, $i = -15 \ldots, 15$.

4.1 Discovering the Structure of Natural Image Patches

Unsupervised learning was first used for discovering the underlying structure of natural image patches by Olshausen and Field [23]. Without making any a priori assumption about the data except a parsimony principle, the method is able to produce small prototypes that resemble Gabor wavelets—that is, spatially localized oriented basis functions. The results were found impressive by the scientific community and their work received substantial attention. It is also known that such results can also be achieved with CNNs [25]. We show in this section that this is also the case for convolutional kernel networks, even though they are not explicitly trained to reconstruct data.

Following [23], we randomly select a database of $300\,000$ whitened natural image patches of size 12×12 and learn $p = 256$ filters $\mathbf{W}$ using the formulation (4). We initialize $\mathbf{W}$ with Gaussian random noise without performing the K-means step, in order to ensure that the output we obtain is not an artifact of the initialization. In Figure 3, we display the filters associated to the top-128 largest weights η_l. Among the 256 filters, 197 exhibit interpretable Gabor-like structures and the rest was less interpretable. To the best of our knowledge, this is the first time that the explicit kernel map of the Gaussian kernel for whitened natural image patches is shown to be related to Gabor wavelets.

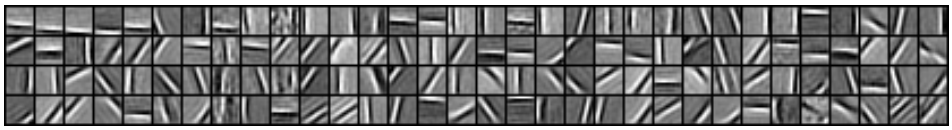


Figure 3: Filters obtained by the first layer of the convolutional kernel network on natural images.

Tr. size	CNN [25]	Scat-1 [8]	Scat-2 [8]	CKN-GM1 $(12/50)$	CKN-GM2 $(12/400)$	CKN-PM1 (200)	CKN-PM2 $(50/200)$	[31]	[18]	[19]
300	7.18	4.7	5.6	4.39	4.24	5.98	**4.15**	NA		
$1K$	3.21	2.3	2.6	2.60	**2.05**	3.23	2.76	NA		
$2K$	2.53	**1.3**	1.8	1.85	1.51	1.97	2.28	NA		
$5K$	1.52	**1.03**	1.4	1.41	1.21	1.41	1.56	NA		
$10K$	0.85	**0.88**	1	1.17	**0.88**	1.18	1.10	NA		
$20K$	0.76	0.79	**0.58**	0.89	0.60	0.83	0.77	NA		
$40K$	0.65	0.74	0.53	0.68	**0.51**	0.64	0.58	NA		
$60K$	0.53	0.70	0.4	0.58	**0.39**	0.63	0.53	0.47	0.45	0.53

Table 1: Test error in $\%$ for various approaches on the MNIST dataset without data augmentation. The numbers in parentheses represent the size p_1 and p_2 of the feature maps at each layer.

4.2 Digit Classification on MNIST

The MNIST dataset [22] consists of $60\,000$ images of handwritten digits for training and $10\,000$ for testing. We use two types of initial maps in our networks: the "patch map", denoted by CNK-PM and the "gradient map", denoted by CNK-GM. We follow the evaluation methodology of [25] for comparison when varying the training set size. We follow Section 3.4 for building the networks. We select the SVM parameter by 5-fold cross validation when the training size is smaller than $20\,000$, or otherwise, we keep $10\,0000$ examples from the training set for validation. We report in Table 1 the results obtained for four simple architectures. CKN-GM1 is the simplest one: its second layer uses 3×3 patches and only $p_2 = 50$ filters, resulting in a network with $5\,400$ parameters. Yet, it achieves an outstanding performance of 0.58% error on the full dataset. The best performing, CKN-GM2, is similar to CKN-GM1 but uses $p_2 = 400$ filters. When working with raw patches, two layers (CKN-PM2) gives better results than one layer. More details about the network architectures are provided in the supplementary material. In general, our method achieves a state-of-the-art accuracy for this task since lower error rates have only been reported by using data augmentation [11].

4.3 Visual Recognition on CIFAR-10 and STL-10

We now move to the more challenging datasets CIFAR-10 [20] and STL-10 [13]. We again follow Section 3.4 for choosing all parameters. We select the best architectures on a validation set of $10\,000$ examples from the training set for CIFAR-10, and by 5-fold cross-validation on STL-10 since the latter only contains 100 training examples per class. We report the results for CKN-GM, defined in the previous section, without exploiting color information, and CKN-PM when working on raw RGB patches whose mean color is subtracted. The best selected models have always two layers. with 800 filters for the top layer. Since CKN-PM and CKN-GM exploit a different information, we also report a combination of such two models, CKN-CO, by concatenating normalized image representations together. The standard deviations for STL-10, obtained using the 10 training folds, was always below 0.7% for our results. Our approach appears to be competitive with the state of the art, especially on STL-10 where only one method does better than ours, despite the fact that our models only use 2 layers and require learning few parameters (see supplementary material).

Method	[12]	[27]	[18]	[13]	[4]	[17]	[31]	CKN-GM	CKN-PM	CKN-CO
CIFAR-10	82.0	82.2	**88.32**	79.6	NA	83.96	84.87	74.84	78.30	82.18
STL-10	60.1	58.7	NA	51.5	**64.5**	62.3	NA	60.04	60.25	62.32

Table 2: Accuracy in $\%$ obtained on the CIFAR-10 and STL-10 datasets without data augmentation.

5 Conclusion

In this paper, we have proposed a new methodology for combining kernels and convolutional neural networks. We show that mixing the ideas of these two concepts is fruitful, since we achieve near state-of-the-art performance on several datasets such as MNIST, CIFAR-10, and STL10, with simple architectures and no data augmentation. Some challenges regarding our work are left open for the future. The first one is the use of supervision to better approximate the kernel for the prediction task. The second consists in leveraging the kernel interpretation of our convolutional neural networks to better understand the theoretical properties of the feature spaces that these networks produce.

References

[1] Y. Bengio. Learning deep architectures for AI. *Found. Trends Mach. Learn.*, 2009.

[2] L. Bo, K. Lai, X. Ren, and D. Fox. Object recognition with hierarchical kernel descriptors. In *Proc. CVPR*, 2011.

[3] L. Bo, X. Ren, and D. Fox. Kernel descriptors for visual recognition. In *Adv. NIPS*, 2010.

[4] L. Bo, X. Ren, and D. Fox. Unsupervised feature learning for RGB-D based object recognition. In *Experimental Robotics*, 2013.

[5] L. Bo and C. Sminchisescu. Efficient match kernel between sets of features for visual recognition. In *Adv. NIPS*, 2009.

[6] L. Bottou, O. Chapelle, D. DeCoste, and J. Weston. *Large-Scale Kernel Machines (Neural Information Processing)*. The MIT Press, 2007.

[7] J. V. Bouvrie, L. Rosasco, and T. Poggio. On invariance in hierarchical models. In *Adv. NIPS*, 2009.

[8] J. Bruna and S. Mallat. Invariant scattering convolution networks. *IEEE T. Pattern Anal.*, 35(8):1872–1886, 2013.

[9] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, 1995.

[10] Y. Cho and L. K. Saul. Large-margin classification in infinite neural networks. *Neural Comput.*, 22(10), 2010.

[11] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Proc. CVPR*, 2012.

[12] A. Coates and A. Y. Ng. Selecting receptive fields in deep networks. In *Adv. NIPS*, 2011.

[13] A. Coates, A. Y. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proc. AISTATS*, 2011.

[14] D. Decoste and B. Schölkopf. Training invariant support vector machines. *Mach. Learn.*, 46(1-3):161–190, 2002.

[15] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. *preprint arXiv:1310.1531*, 2013.

[16] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, 2008.

[17] R. Gens and P. Domingos. Discriminative learning of sum-product networks. In *Adv. NIPS*, 2012.

[18] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *Proc. ICML*, 2013.

[19] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Proc. ICCV*, 2009.

[20] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Tech. Rep.*, 2009.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Adv. NIPS*, 2012.

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

[23] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[24] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Adv. NIPS*, 2007.

[25] M. Ranzato, F.-J. Huang, Y-L. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. CVPR*, 2007.

[26] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. 2004.

[27] K. Sohn and H. Lee. Learning invariant representations with local transformations. In *Proc. ICML*, 2012.

[28] G. Wahba. *Spline models for observational data*. SIAM, 1990.

[29] L. Wan, M. D. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. In *Proc. ICML*, 2013.

[30] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Adv. NIPS*, 2001.

[31] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *preprint arXiv:1301.3557*, 2013.

[32] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *preprint arXiv:1311.2901v3*, 2013.

A List of Architectures Reported in the Experiments

We present in details the architectures used in the paper in Table 3.

Arch.	N	m_1	p_1	κ_1	m_2	p_2	S	$\sharp$ param
MNIST								
CKN-GM1	2	1×1	12	0.5	3×3	50	4×4	$5\,400$
CKN-GM2	2	1×1	12	0.5	3×3	400	3×3	$43\,200$
CKN-PM1	1	5×5	200	0.5	-	-	4×4	$5\,000$
CKN-PM2	2	5×5	50	0.5	2×2	200	6×6	$41\,250$
CIFAR-10								
CKN-GM	2	1×1	12	0.5	2×2	800	4×4	$38\,400$
CKN-PM	2	2×2	100	0.5	2×2	800	4×4	$321\,200$
STL-10								
CKN-GM	2	1×1	12	0.5	3×3	800	4×4	$86\,400$
CKN-PM	2	3×3	50	0.5	3×3	800	3×3	$361\,350$

Table 3: List of architectures reported in the paper. N is the number of layers; p_1 and p_2 represent the number of filters for each layer; m_1 and m_2 represent the size of the patches $\mathcal{N}_1$ and $\mathcal{N}_2$ that are of size $m_1 \times m_1$ and $m_2 \times m_2$ on their respective feature maps ζ_1 and ζ_2; κ_1 is the subsampling factor between layer 1 and layer 2; S is the size of the output feature map, and the last column indicates the number of parameters that the network has to learn.