Towards Deep Kernel Machines

Julien Mairal

Inria, Grenoble

Prague, April, 2017
Part I: Scientific Context
A quick zoom on multilayer neural networks

The goal is to learn a prediction function $f : \mathbb{R}^p \to \mathbb{R}$ given labeled training data $(x_i, y_i)_{i=1,...,n}$ with $x_i$ in $\mathbb{R}^p$, and $y_i$ in $\mathbb{R}$:

$$\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \frac{\lambda}{n} \Omega(f).$$

regularization

empirical risk, data fit
A quick zoom on multilayer neural networks

The goal is to learn a **prediction function** $f : \mathbb{R}^p \rightarrow \mathbb{R}$ given labeled training data $(x_i, y_i)_{i=1,...,n}$ with $x_i$ in $\mathbb{R}^p$, and $y_i$ in $\mathbb{R}$:

$$
\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \lambda \Omega(f).
$$

What is specific to multilayer neural networks?

- The “neural network” space $\mathcal{F}$ is explicitly parametrized by:

  $$
  f(x) = \sigma_k(A_k \sigma_{k-1}(A_{k-1} \ldots \sigma_2(A_2 \sigma_1(A_1 x)) \ldots)).
  $$

- Finding the optimal $A_1, A_2, \ldots, A_k$ yields a **non-convex** optimization problem in **huge dimension**.
A quick zoom on convolutional neural networks

Figure: Picture from LeCun et al. [1998]

- CNNs perform “simple” operations such as convolutions, pointwise non-linearities and subsampling.
- For most successful applications of CNNs, training is supervised.
A quick zoom on convolutional neural networks

What are the main features of CNNs?
- they capture *compositional* and *multiscale* structures in images;
- they provide some *invariance*;
- they model *local stationarity* of images at several scales.

What are the main open problems?
- very little theoretical understanding;
- they require large amounts of labeled data;
- they require manual design and parameter tuning;

Nonetheless...
- they are the focus of a huge academic and industrial effort;
- there is efficient and well-documented open-source software.
A quick zoom on convolutional neural networks

What are the main features of CNNs?
- they capture **compositional** and **multiscale** structures in images;
- they provide some **invariance**;
- they model **local stationarity** of images at several scales.

What are the main open problems?
- very little **theoretical understanding**;
- they require **large amounts of labeled data**;
- they require **manual design and parameter tuning**;

Nonetheless...

...they are the focus of a huge academic and industrial effort; there is efficient and well-documented open-source software.
A quick zoom on convolutional neural networks

What are the main features of CNNs?
- they capture **compositional** and **multiscale** structures in images;
- they provide some **invariance**;
- they model **local stationarity** of images at several scales.

What are the main open problems?
- very little **theoretical understanding**;
- they require **large amounts of labeled data**;
- they require **manual design and parameter tuning**;

Nonetheless...
- they are the focus of a **huge academic and industrial effort**;
- there is **efficient and well-documented open-source software**.
Context of kernel methods

\[ \phi(S) = (aatcgagtcac, atggacgtct, tgcactact) \]

\[
\begin{pmatrix}
1 & 0.5 & 0.3 \\
0.5 & 1 & 0.6 \\
0.3 & 0.6 & 1 \\
\end{pmatrix}
\]

Idea: representation by pairwise comparisons

- Define a “comparison function”: \( K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \).
- Represent a set of \( n \) data points \( S = \{x_1, \ldots, x_n\} \) by the \( n \times n \) matrix:
  \[
  K_{ij} := K(x_i, x_j).
  \]

Context of kernel methods

Theorem (Aronszajn, 1950)

\[ K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \text{ is a positive definite kernel if and only if there exists a} \]

Hilbert space \( \mathcal{H} \) and a mapping

\[ \varphi : \mathcal{X} \rightarrow \mathcal{H}, \]

such that, for any \( x, x' \) in \( \mathcal{X} \),

\[ K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}. \]
Context of kernel methods

The classical challenge of kernel methods

Find a kernel $K$ such that

- the data in the feature space $\mathcal{H}$ has **nice properties**, e.g., linear separability, cluster structure.
- $K$ is **fast to compute** and **mathematically valid** (p.d.).
Context of kernel methods (supervised learning)

The goal is to learn a prediction function \( f : \mathcal{X} \to \mathbb{R} \) given labeled training data \((x_i, y_i)_{i=1,...,n}\) with \(x_i\) in \(\mathcal{X}\), and \(y_i\) in \(\mathbb{R}\):

\[
\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \lambda \|f\|^2_{\mathcal{H}}.
\]

What is specific here to kernel methods?

- The “kernel method” space \(\mathcal{H}\) is possibly infinite-dimensional.
- Optimization over \(f\) is done implicitly by (often) minimizing a convex function.
- \(\mathcal{X}\) does not need to be a vector space.
Context of kernel methods

What are the main features of kernel methods?

- **decoupling** of data representation and learning algorithm;
- a huge number of **unsupervised and supervised** algorithms;
- typically, **convex optimization problems** in a supervised context;
- **versatility**: applies to vectors, sequences, graphs, sets, . . . ;
- **natural regularization function** to control the learning capacity;
- **well studied theoretical framework**.
Context of kernel methods

What are the main features of kernel methods?

- **decoupling** of data representation and learning algorithm;
- a huge number of *unsupervised and supervised* algorithms;
- typically, **convex optimization problems** in a supervised context;
- **versatility**: applies to vectors, sequences, graphs, sets, . . . ;
- **natural regularization function** to control the learning capacity;
- well studied theoretical framework.

But...

- **poor scalability in** $n$, at least $O(n^2)$;
- **decoupling** of data representation and learning may not be a good thing, according to recent *supervised* deep learning success.
Context of kernel methods

Challenges

- **Scaling-up kernel methods** with approximate feature maps;

\[ K(x, x') \approx \langle \psi(x), \psi(x') \rangle. \]


- Design **data-adaptive and task-adaptive** kernels;
- Build **kernel hierarchies** to capture **compositional** structures.
- Introduce **supervision** in the kernel design.
Context of kernel methods

Challenges

- **Scaling-up kernel methods** with approximate feature maps;

\[ K(x, x') \approx \langle \psi(x), \psi(x') \rangle. \]


- Design **data-adaptive and task-adaptive** kernels;
- Build **kernel hierarchies** to capture **compositional** structures.
- Introduce **supervision** in the kernel design.

**We need deep kernel machines!**
Some more motivation

Longer term objectives

- build a kernel for images (abstract object), for which we can precisely quantify the **invariance, stability to perturbations, recovery, and complexity** properties.
- build deep networks which can be easily **regularized**.
- build deep networks for **structured objects** (graph, sequences)...
- add more **geometric interpretation** to deep networks.
- ...
Part II: Basic Principles of Deep Kernel Machines
Basic principles of deep kernel machines: composition

Composition of feature spaces

Consider a p.d. kernel $K_1 : \mathcal{X}^2 \to \mathbb{R}$ and its RKHS $\mathcal{H}_1$ with mapping $\varphi_1 : \mathcal{X} \to \mathcal{H}_1$. Consider also a p.d. kernel $K_2 : \mathcal{H}_1^2 \to \mathbb{R}$ and its RKHS $\mathcal{H}_2$ with mapping $\varphi_2 : \mathcal{H}_1 \to \mathcal{H}_2$. Then, $K_3 : \mathcal{X}^2 \to \mathbb{R}$ below is also p.d.

$$K_3(x, x') = K_2(\varphi_1(x), \varphi_1(x')),$$

and its RKHS mapping is $\varphi_3 = \varphi_2 \circ \varphi_1$. 
Basic principles of deep kernel machines: composition

Composition of feature spaces

Consider a p.d. kernel \( K_1 : \mathcal{X}^2 \to \mathbb{R} \) and its RKHS \( \mathcal{H}_1 \) with mapping \( \varphi_1 : \mathcal{X} \to \mathcal{H}_1 \). Consider also a p.d. kernel \( K_2 : \mathcal{H}_1^2 \to \mathbb{R} \) and its RKHS \( \mathcal{H}_2 \) with mapping \( \varphi_2 : \mathcal{H}_1 \to \mathcal{H}_2 \). Then, \( K_3 : \mathcal{X}^2 \to \mathbb{R} \) below is also p.d.

\[
K_3(x, x') = K_2(\varphi_1(x), \varphi_1(x')),
\]

and its RKHS mapping is \( \varphi_3 = \varphi_2 \circ \varphi_1 \).

Examples

\[
K_3(x, x') = e^{-\frac{1}{2\sigma^2} \| \varphi_1(x) - \varphi_1(x') \|_{\mathcal{H}_1}^2}.
\]

\[
K_3(x, x') = \langle \varphi_1(x), \varphi_1(x') \rangle_{\mathcal{H}_1}^2 = K_1(x, x')^2.
\]
Basic principles of deep kernel machines: composition

Remarks on the composition of feature spaces

- we can iterate the process many times.
- the idea appears early in the literature of kernel methods [see Schölkopf et al., 1998, for a multilayer variant of kernel PCA].

Is this idea sufficient to make kernel methods more powerful?
Basic principles of deep kernel machines: composition

Remarks on the composition of feature spaces

- we can iterate the process many times.
- the idea appears early in the literature of kernel methods [see Schölkopf et al., 1998, for a multilayer variant of kernel PCA].

Is this idea sufficient to make kernel methods more powerful?

Probably not:

- $K_2$ is doomed to be a simple kernel (dot-product or RBF kernel).
- it does not address any of previous challenges.
- $K_3$ and $K_1$ operate on the same type of object; it is not clear why designing $K_3$ is easier than designing $K_1$ directly.
Basic principles of deep kernel machines: composition

Remarks on the composition of feature spaces

- we can iterate the process many times.
- the idea appears early in the literature of kernel methods [see Schölkopf et al., 1998, for a multilayer variant of kernel PCA].

Is this idea sufficient to make kernel methods more powerful?

Probably not:

- $K_2$ is doomed to be a simple kernel (dot-product or RBF kernel).
- it does not address any of previous challenges.
- $K_3$ and $K_1$ operate on the same type of object; it is not clear why designing $K_3$ is easier than designing $K_1$ directly.

Nonetheless, we will see later that this idea can be used to build a hierarchies of kernels that operate on more and more complex objects.
Basic principles of deep kernel machines: infinite NN

A large class of kernels on $\mathbb{R}^p$ may be defined as an expectation

$$K(x, y) = \mathbb{E}_w[s(w^\top x)s(w^\top y)],$$

where $s : \mathbb{R} \to \mathbb{R}$ is a nonlinear function. The encoding can be seen as a one-layer neural network with infinite number of random weights.
Basic principles of deep kernel machines: infinite NN

A large class of kernels on $\mathbb{R}^p$ may be defined as an expectation

$$K(x, y) = \mathbb{E}_w [s(w^\top x) s(w^\top y)],$$

where $s : \mathbb{R} \to \mathbb{R}$ is a nonlinear function. The encoding can be seen as a one-layer neural network with infinite number of random weights.

Examples

- random Fourier features

$$\kappa(x - y) = \mathbb{E}_{w \sim q(w), b \sim \mathcal{U}[0, 2\pi]} \left[ \sqrt{2} \cos(w^\top x + b) \sqrt{2} \cos(w^\top y + b) \right]$$

- Gaussian kernel

$$e^{-\frac{1}{2\sigma^2} \|x - y\|_2^2} \propto \mathbb{E}_w \left[ e^{\frac{2}{\sigma^2} w^\top x} e^{\frac{2}{\sigma^2} w^\top y} \right] \quad \text{with} \quad w \sim \mathcal{N}(0, (\sigma^2/4)I).$$
Basic principles of deep kernel machines: infinite NN

Example, arc-cosine kernels

\[ K(x, y) \propto \mathbb{E}_w \left[ \max \left( w^\top x, 0 \right)^\alpha \max \left( w^\top y, 0 \right)^\alpha \right] \quad \text{with} \quad w \sim \mathcal{N}(0, I), \]

for \( x, y \) on the hyper-sphere \( S^{m-1} \). Interestingly, the non-linearity \( s \) are typical ones from the neural network literature.

- \( s(u) = \max(0, u) \) (rectified linear units) leads to
  \[ K_1(x, y) = \sin(\theta) + (\pi - \theta) \cos(\theta) \quad \text{with} \quad \theta = \cos^{-1}(x^\top y); \]
- \( s(u) = \max(0, u)^2 \) (squared rectified linear units) leads to
  \[ K_2(x, y) = 3 \sin(\theta) \cos(\theta) + (\pi - \theta)(1 + 2 \cos^2(\theta)); \]

Remarks

- infinite neural nets were discovered by Neal, 1994; then revisited many times [Le Roux, 2007, Cho and Saul, 2009].
- the concept does not lead to more powerful kernel methods...
Basic principles of DKM: dot-product kernels

Another basic link between kernels and neural networks can be obtained by considering dot-product kernels.

A classical old result

Let $\mathcal{X} = \mathbb{S}^{d-1}$ be the unit sphere of $\mathbb{R}^d$. The kernel $K : \mathcal{X}^2 \to \mathbb{R}$

$$K(x, y) = \kappa(\langle x, y \rangle_{\mathbb{R}^d})$$

is positive definite if and only if $\kappa$ is smooth and its Taylor expansion coefficients are non-negative.

Remark

- the proposition holds if $\mathcal{X}$ is the unit sphere of some Hilbert space and $\langle x, y \rangle_{\mathbb{R}^d}$ is replaced by the corresponding inner-product.
Basic principles of DKM: dot-product kernels

The Nyström method consists of replacing any point \( \varphi(x) \) in \( \mathcal{H} \), for \( x \) in \( \mathcal{X} \) by its orthogonal projection onto a finite-dimensional subspace

\[
\mathcal{F} = \text{span}(\varphi(z_1), \ldots, \varphi(z_p)),
\]

for some anchor points \( Z = [z_1, \ldots, z_p] \) in \( \mathbb{R}^{d \times p} \).
Basic principles of DKM: dot-product kernels

The projection is equivalent to

$$\Pi_F[x] := \sum_{j=1}^{p} \beta_j^* \varphi(z_j) \text{ with } \beta^* \in \arg\min_{\beta \in \mathbb{R}^p} \| \varphi(x) - \sum_{j=1}^{p} \beta_j \varphi(z_j) \|_2^2$$

Then, it is possible to show that with $K(x, y) = \kappa(\langle x, y \rangle_{\mathbb{R}^d})$,

$$K(x, y) \approx \langle \Pi_F[x], \Pi_F[y] \rangle_{\mathcal{H}} = \langle \psi(x), \psi(y) \rangle_{\mathbb{R}^p},$$

with

$$\psi(x) = \kappa(Z^\top Z)^{-1/2} \kappa(Z^\top x),$$

where the function $\kappa$ is applied pointwise to its arguments. The resulting $\psi$ can be interpreted as a neural network performing (i) linear operation, (ii) pointwise non-linearity, (iii) linear operation.
Part III: Convolutional Kernel Networks
Convolutional kernel networks

The (happy?) marriage of kernel methods and CNNs

1. **a multilayer convolutional kernel for images**: A hierarchy of kernels for local image neighborhoods (aka, receptive fields).

2. **unsupervised scheme for large-scale learning**: the kernel being too computationally expensive, the Nyström approximation at each layer yields a new type of unsupervised deep neural network.

3. **end-to-end learning**: learning subspaces in the RKHSs can be achieved with a supervised loss function.
Convolutional kernel networks

The (happy?) marriage of kernel methods and CNNs

1. a multilayer convolutional kernel for images: A hierarchy of kernels for local image neighborhoods (aka, receptive fields).

2. unsupervised scheme for large-scale learning: the kernel being too computationally expensive, the Nyström approximation at each layer yields a new type of unsupervised deep neural network.

3. end-to-end learning: learning subspaces in the RKHSs can be achieved with a supervised loss function.

First proof of concept with unsupervised learning


The model of this presentation

Related work

- proof of concept for combining kernels and deep learning [Cho and Saul, 2009];
- hierarchical kernel descriptors [Bo et al., 2011];
- other multilayer models [Bouvrie et al., 2009, Montavon et al., 2011, Anselmi et al., 2015];
- deep Gaussian processes [Damianou and Lawrence, 2013].
- multilayer PCA [Schölkopf et al., 1998].
- old kernels for images [Scholkopf, 1997].
- RBF networks [Broomhead and Lowe, 1988].
The multilayer convolutional kernel

Definition: image feature maps

An image feature map is a function $I : \Omega \rightarrow \mathcal{H}$, where $\Omega$ is a 2D grid representing “coordinates” in the image and $\mathcal{H}$ is a Hilbert space.
The multilayer convolutional kernel

Definition: image feature maps

An image feature map is a function $I : \Omega \rightarrow \mathcal{H}$, where $\Omega$ is a 2D grid representing “coordinates” in the image and $\mathcal{H}$ is a Hilbert space.

Motivation and examples

- Each point $I(\omega)$ carries information about an image neighborhood, which is motivated by the **local stationarity** of natural images.
- We will construct a sequence of maps $I_0, \ldots, I_k$. Going up in the hierarchy yields **larger receptive fields** with **more invariance**.
- $I_0$ may simply be the input image, where $\mathcal{H}_0 = \mathbb{R}^3$ for RGB.
The multilayer convolutional kernel

Definition: image feature maps
An image feature map is a function $I : \Omega \rightarrow \mathcal{H}$, where $\Omega$ is a 2D grid representing “coordinates” in the image and $\mathcal{H}$ is a Hilbert space.

Motivation and examples
- Each point $I(\omega)$ carries information about an image neighborhood, which is motivated by the **local stationarity** of natural images.
- We will construct a sequence of maps $I_0, \ldots, I_k$. Going up in the hierarchy yields **larger receptive fields** with **more invariance**.
- $I_0$ may simply be the input image, where $\mathcal{H}_0 = \mathbb{R}^3$ for RGB.

How do we go from $I_0 : \Omega_0 \rightarrow \mathcal{H}_0$ to $I_1 : \Omega_1 \rightarrow \mathcal{H}_1$?
The multilayer convolutional kernel

Definition: image feature maps

An image feature map is a function $I : \Omega \rightarrow \mathcal{H}$, where $\Omega$ is a 2D grid representing “coordinates” in the image and $\mathcal{H}$ is a Hilbert space.

Motivation and examples

- Each point $I(\omega)$ carries information about an image neighborhood, which is motivated by the local stationarity of natural images.
- We will construct a sequence of maps $I_0, \ldots, I_k$. Going up in the hierarchy yields larger receptive fields with more invariance.
- $I_0$ may simply be the input image, where $\mathcal{H}_0 = \mathbb{R}^3$ for RGB.

How do we go from $I_0 : \Omega_0 \rightarrow \mathcal{H}_0$ to $I_1 : \Omega_1 \rightarrow \mathcal{H}_1$?

First, define a p.d. kernel on patches of $I_0$!
The multilayer convolutional kernel

Going from \( l_0 \) to \( l_{0.5} \): kernel trick

- Patches of size \( e_0 \times e_0 \) can be defined as elements of the \textbf{Cartesian product} \( \mathcal{P}_0 := \mathcal{H}^{e_0 \times e_0} \) endowed with its natural inner-product.
- \textbf{Define a p.d. kernel on such patches}: For all \( x, x' \) in \( \mathcal{P}_0 \),

\[
K_1(x, x') = \|x\|_{\mathcal{P}_0} \|x'\|_{\mathcal{P}_0} \kappa_1 \left( \frac{\langle x, x' \rangle_{\mathcal{P}_0}}{\|x\|_{\mathcal{P}_0} \|x'\|_{\mathcal{P}_0}} \right) \quad \text{if} \; x, x' \neq 0 \; \text{and} \; 0 \text{ otherwise.}
\]

Note that for \( y, y' \) normalized, we may choose

\[
\kappa_1 (\langle y, y' \rangle_{\mathcal{P}_0}) = e^{\alpha_1 (\langle y, y' \rangle_{\mathcal{P}_0} - 1)} = e^{-\frac{\alpha_1}{2} \|y-y'\|_{\mathcal{P}_0}^2}.
\]

- \textbf{We call} \( \mathcal{H}_1 \) the RKHS and define a \textbf{mapping} \( \varphi_1 : \mathcal{P}_0 \rightarrow \mathcal{H}_1 \).
- Then, we may define the map \( l_{0.5} : \Omega_0 \rightarrow \mathcal{H}_1 \) that carries the representations in \( \mathcal{H}_1 \) of the patches from \( l_0 \) at all locations in \( \Omega_0 \).
The multilayer convolutional kernel

How do we go from $I_{0.5} : \Omega_0 \rightarrow \mathcal{H}_1$ to $I_1 : \Omega_1 \rightarrow \mathcal{H}_1$?
The multilayer convolutional kernel

How do we go from \( I_{0.5} : \Omega_0 \to \mathcal{H}_1 \) to \( I_1 : \Omega_1 \to \mathcal{H}_1 \)?

**Linear pooling!**
The multilayer convolutional kernel

Going from $I_{0.5}$ to $I_1$: linear pooling

- For all $\omega$ in $\Omega_1$:

$$I_1(\omega) = \sum_{\omega' \in \Omega_0} I_{0.5}(\omega') e^{-\beta_1 \| \omega' - \omega \|^2}.$$ 

- The Gaussian weight can be interpreted as an anti-aliasing filter for downsampling the map $I_{0.5}$ to a different resolution.

- Linear pooling is compatible with the kernel interpretation: linear combinations of points in the RKHS are still points in the RKHS.

Finally,

- We may now repeat the process and build $I_0, I_1, \ldots, I_k$.

- and obtain the multilayer convolutional kernel

$$K(I_k, I'_k) = \sum_{\omega \in \Omega_k} \langle I_k(\omega), I'_k(\omega) \rangle_{\mathcal{H}_k}.$$
The multilayer convolutional kernel

In summary

- The multilayer convolutional kernel builds upon similar principles as a convolutional neural net (*multiscale, local stationarity*).
- Invariance to local translations is achieved through **linear pooling** in the RKHS.
- It remains a **conceptual object** due to its high complexity.
- **Learning and modelling are still decoupled.**

Let us first address the second point (scalability).
Unsupervised learning for convolutional kernel networks

Learn linear subspaces of finite-dimensions where we project the data

Figure: The convolutional kernel network model between layers 0 and 1.
Unsupervised learning for convolutional kernel networks

Formally, this means using the Nyström approximation

- We now manipulate **finite-dimensional maps** \( M_j : \Omega_j \rightarrow \mathbb{R}^{p_j} \).
- Every linear subspace is parametrized by anchor points

\[
\mathcal{F}_j := \text{Span} \left( \varphi(z_{j,1}), \ldots, \varphi(z_{j,p_j}) \right),
\]

where the \( z_{1,j} \)'s are in \( \mathbb{R}^{p_{j-1}e_{j-1}} \) for patches of size \( e_{j-1} \times e_{j-1} \).
- The encoding function at layer \( j \) is

\[
\psi_j(x) := \|x\| \kappa_j(Z_j^\top Z_j)^{-1/2} \kappa_1 \left( Z_j^\top \frac{x}{\|x\|} \right) \text{ if } x \neq 0 \text{ and } 0 \text{ otherwise},
\]

where \( Z_j = [z_{j,1}, \ldots, z_{j,p_j}] \) and \( \| . \| \) is the Euclidean norm.
- The interpretation is **convolution** with filters \( Z_j \), **pointwise non-linearity**, \( 1 \times 1 \) **convolution**, **contrast normalization**.
Unsupervised learning for convolutional kernel networks

- The pooling operation keeps points in the linear subspace $\mathcal{F}_j$, and pooling $M_{0.5} : \Omega_0 \rightarrow \mathbb{R}^{p_1}$ is equivalent to pooling $I_{0.5} : \Omega_0 \rightarrow \mathcal{H}_1$.

Figure: The convolutional kernel network model between layers 0 and 1.
Unsupervised learning for convolutional kernel networks

How do we learn the filters with no supervision?

we learn one layer at a time, starting from the bottom one.

- we extract a large number—say 100 000 patches from layers $j - 1$
  computed on an image database and normalize them;
- perform a spherical K-means algorithm to learn the filters $Z_j$;
- compute the projection matrix $\kappa_j(Z_j^\top Z_j)^{-1/2}$.

Remarks

- with kernels, we map patches in infinite dimension; with the
  projection, we manipulate finite-dimensional objects.
- we obtain an unsupervised convolutional net with a geometric
  interpretation, where we perform projections in the RKHSs.
Unsupervised learning for convolutional kernel networks

Remark on input image pre-processing

Unsupervised CKNs are sensitive to pre-processing; we have tested

- RAW RGB input;
- local centering of every color channel;
- local whitening of each color channel;
- 2D image gradients.

(a) RAW RGB

(b) centering
Unsupervised learning for convolutional kernel networks

Remark on input image pre-processing

Unsupervised CKNs are sensitive to pre-processing; we have tested

- RAW RGB input;
- local \textit{centering} of every color channel;
- local \textit{whitening} of each color channel;
- 2D \textit{image gradients}.

(c) RAW RGB \hspace{1cm} (d) whitening
Unsupervised learning for convolutional kernel networks

Remark on pre-processing with image gradients and $1 \times 1$ patches

- Every pixel/patch can be represented as a two dimensional vector
  \[ \mathbf{x} = \rho [\cos(\theta), \sin(\theta)], \]
  where $\rho = \|\mathbf{x}\|$ is the gradient intensity and $\theta$ is the orientation.

- A natural choice of filters $\mathbf{Z}$ would be
  \[ \mathbf{z}_j = [\cos(\theta_j), \sin(\theta_j)] \text{ with } \theta_j = 2j\pi/p_0. \]

- Then, the vector $\psi(\mathbf{x}) = \|\mathbf{x}\|\kappa_1(\mathbf{Z}^\top \mathbf{Z})^{-1/2}\kappa_1 \left( \mathbf{Z}^\top \frac{\mathbf{x}}{\|\mathbf{x}\|} \right)$, can be interpreted as a "soft-binning" of the gradient orientation.

- After pooling, the representation of this first layer is very close to SIFT/HOG descriptors [see Bo et al., 2011].
Convolutional kernel networks with supervised learning

How do we learn the filters with supervision?

- Given a kernel $K$ and RKHS $\mathcal{H}$, the ERM objective is

$$
\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2.
$$

- empirical risk, data fit

- regularization

- here, we use the parametrized kernel

$$
K_Z(l_0, l_0') = \sum_{\omega \in \Omega_k} \langle M_k(\omega), M'_k(\omega) \rangle = \langle M_k, M'_k \rangle_F,
$$

- and we obtain the simple formulation

$$
\min_{W \in \mathbb{R}^{p_k \times |\Omega_k|}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, \langle W, M_k^i \rangle_F) + \frac{\lambda}{2} \|W\|_F^2. \tag{1}
$$
Convolutional kernel networks with supervised learning

How do we learn the filters with supervision?

- we **jointly optimize** w.r.t. $\mathcal{Z}$ (set of filters) and $\mathbf{W}$.
- we **alternate** between the optimization of $\mathcal{Z}$ and of $\mathbf{W}$;
- for $\mathbf{W}$, the problem is strongly-convex and can be tackled with recent algorithms that are much faster than SGD;
- for $\mathcal{Z}$, we derive **backpropagation rules** and use classical tricks for learning CNNs (SGD+$\text{momentum}$);

The only tricky part is to differentiate $\kappa_j(\mathbf{Z}_j^\top\mathbf{Z}_j)^{-1/2}$ w.r.t $\mathbf{Z}_j$, which is a non-standard operation in classical CNNs.
Convolutional kernel networks

In summary

- a multilayer kernel for images, which builds upon similar principles as a convolutional neural net (multiscale, local stationarity).
- A new type of convolutional neural network with a geometric interpretation: orthogonal projections in RKHS.
- Learning may be unsupervised: align subspaces with data.
- Learning may be supervised: subspace learning in RKHSs.
Part IV: Applications
Image classification

Experiments were conducted on classical “deep learning” datasets, on CPUs with no model averaging and no data augmentation.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># classes</th>
<th>im. size</th>
<th>n_{train}</th>
<th>n_{test}</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>10</td>
<td>32 × 32</td>
<td>50 000</td>
<td>10 000</td>
</tr>
<tr>
<td>SVHN</td>
<td>10</td>
<td>32 × 32</td>
<td>604 388</td>
<td>26 032</td>
</tr>
</tbody>
</table>

Figure: Figure from the NIPS’16 paper. Error rates in percents.

Remarks on CIFAR-10

- 10% is the standard “good” result for single model with no data augmentation.
- the best unsupervised architecture has two layers, is wide (1024-16384 filters), and achieves 14.2%;
Image super-resolution

The task is to predict a high-resolution $y$ image from low-resolution one $x$. This may be formulated as a **multivariate regression problem**.

(a) Low-resolution $y$

(b) High-resolution $x$
Image super-resolution

The task is to predict a high-resolution $y$ image from low-resolution one $x$. This may be formulated as a **multivariate regression problem**.

(c) Low-resolution $y$  
(d) Bicubic interpolation
Image super-resolution

<table>
<thead>
<tr>
<th>Fact.</th>
<th>Dataset</th>
<th>Bicubic</th>
<th>SC</th>
<th>CNN</th>
<th>CSCN</th>
<th>SCKN</th>
</tr>
</thead>
<tbody>
<tr>
<td>x2</td>
<td>Set5</td>
<td>33.66</td>
<td>35.78</td>
<td>36.66</td>
<td>36.93</td>
<td>37.07</td>
</tr>
<tr>
<td></td>
<td>Set14</td>
<td>30.23</td>
<td>31.80</td>
<td>32.45</td>
<td>32.56</td>
<td>32.76</td>
</tr>
<tr>
<td></td>
<td>Kodim</td>
<td>30.84</td>
<td>32.19</td>
<td>32.80</td>
<td>32.94</td>
<td>33.21</td>
</tr>
<tr>
<td>x3</td>
<td>Set5</td>
<td>30.39</td>
<td>31.90</td>
<td>32.75</td>
<td><strong>33.10</strong></td>
<td>33.08</td>
</tr>
<tr>
<td></td>
<td>Set14</td>
<td>27.54</td>
<td>28.67</td>
<td>29.29</td>
<td>29.41</td>
<td><strong>29.50</strong></td>
</tr>
<tr>
<td></td>
<td>Kodim</td>
<td>28.43</td>
<td>29.21</td>
<td>29.64</td>
<td>29.76</td>
<td><strong>29.88</strong></td>
</tr>
</tbody>
</table>

**Table:** Reconstruction accuracy for super-resolution in PSNR (the higher, the better). All CNN approaches are without data augmentation at test time.

**Remarks**

- CNN is a “vanilla CNN” [Dong et al., 2016];
- Very recent work does better with very deep CNNs and residual learning [Kim et al., 2016];
- CSCN combines ideas from sparse coding and CNNs;

[Zeyde et al., 2010, Dong et al., 2016, Wang et al., 2015, Kim et al., 2016].
Image super-resolution

Figure: Results for x3 upscaling.
Image super-resolution

**Figure:** Bicubic
Image super-resolution

Figure: SCKN
Image super-resolution

Figure: Results for x3 upscaling.
Image super-resolution

Figure: Bicubic
Image super-resolution

Figure: SCKN
Image super-resolution

Figure: Results for x3 upscaling.
Image super-resolution

Figure: Bicubic
Image super-resolution

Figure: SCKN
Image super-resolution

Figure: Results for x3 upscaling.
Image super-resolution

Figure: Bicubic
Image super-resolution

Figure: SCKN


References II


References III


References IV


