Learning a Compact Image Code for Efficient Recognition of Novel Classes

Lorenzo Torresani

Joint work with:
- Alessandro Bergamo (Dartmouth)
- Andrew Fitzgibbon (Microsoft Research Cambridge)
Problem statement: novel-class search

• **Given:**
  - large image database (e.g., 10 million photos)
  - user-provided images of an object class

• **Want:**
  - database images of this class
  - no text/tags available
  - query images may represent a novel class
Big Image Data

- 72 hours of video are uploaded to YouTube every minute
- More than 120M distinct videos
- 6 billion new uploads each month
- 100 billion photos
- 72 hours of video are uploaded to YouTube every minute

Interactive visual search in these collections requires:
- The ability to efficiently train and test novel visual classes at search time
Technical requirements of novel-class search

• The object classifier must be learned on the fly from few examples

• Recognition in the database must have low computational cost

• Image descriptors must be compact to allow storage in memory
State-of-the-art in object classification

Winning recipe: many features + non-linear classifiers (e.g. [Gehler and Nowozin, CVPR’09])

\[ x = \begin{bmatrix} 
\text{GIST} \\
\vdots \\
\text{oriented gradients} \\
\text{SIFT} 
\end{bmatrix} \]

non-linear decision boundary
Multiple kernel combiners for novel-class search?

Classification output is obtained by combining many features via non-linear kernels (e.g. the LP-β of [Gehler and Nowozin, CVPR’09]):

\[ h(x) = \sum_{f=1}^{F} \beta_f \sum_{n=1}^{N} k_f(x, x_n) \alpha_n + b \]

Unsuitable for our needs due to:

- large storage requirements (typically over 20K bytes/image)
- costly evaluation (requires query-time kernel distance computation for each test image)
- costly training (1+ minute for \( O(10) \) training examples)
Our approach [Torresani et al., 2010; Bergamo et al., 2011]

Key-idea: **represent** each image as the binarized output of a large number of **predefined** multiple kernel classifiers

- **Compact**: only $C$ bits per image
- A linear combination of these features is an **efficient** multiple kernel combiner

\[
\Phi(x) = \left[ \begin{array}{c} \phi_1(x) \\ \vdots \\ \phi_C(x) \end{array} \right] \in \{0, 1\}^C
\]

This is the only thing we store in the database

The $c$-th bit is the binarized output of a **pre-learned** LP-$\beta$ for the $c$-th basis class:

\[
\phi_c(x) = 1 \left[ \sum_{f=1}^{F} \beta^c_f \sum_{n=1}^{N} k_f(x, x^c_n) \alpha^c_n \right]
\]

\[
w^T \Phi(x) = \sum_{c=1}^{C} w_c 1 \left[ \sum_{f=1}^{F} \beta^c_f \sum_{n=1}^{N} k_f(x, x^c_n) \alpha^c_n \right]
\]

LP-$\beta$ trained and evaluated **before** the creation of the database
Method overview

1. **Offline learning:**
   training the basis classifiers \( \phi_1, \ldots, \phi_C \)
   defining the compact representation \( \Phi(x) \in \{0, 1\}^C \)

2. **Query-time learning:**
   using the binary code for recognition and retrieval
   training examples of **novel** class

\[
g_{\text{duck}}(\Phi(x)) = \sum_{c=1}^{C} w_{\text{duck}}^c \phi_c(x)
\]

\[\Phi(x_1) \quad \ldots \quad \Phi(x_N)\]
Related work

- Hand-specified visual properties correlated to the classes to recognize
- Used for recognition in specific domains
How do we define the basis classifiers?

- PiCoDes (Picture Codes / Pico-descriptors) [Bergamo et al., 2011]:

we want to choose $\phi_1(x), \ldots, \phi_c(x)$ such that the linear classification model

$$g(\Phi(x); w) = \sum_{c=1}^C w_c \phi_c(x)$$

enables recognition of many classes with good accuracy
The descriptor-learning goal

- **Given:**
  training examples $x_1, \ldots, x_N$, each belonging to one of $K$ classes (where $K$ is large).

- **Want:**
  learn $C$ multiple kernel combiners $\phi_1(x), \ldots, \phi_C(x)$
  s.t. there exist $K$ linear classifiers $(w_1, b_1), \ldots, (w_K, b_K)$
satisfying

  \[
  w_k^T \Phi(x_i) + b_k > 0 \quad \text{if } x_i \text{ belongs to class } k
  \]

  \[
  w_k^T \Phi(x_i) + b_k < 0 \quad \text{otherwise}
  \]
A large margin formulation

- **Training set:**
  \( x_i: \) image \( i \)
  \( y_{i,k} \in \{-1, 1\}: \) label \( y_{i,k} = 1 \) iff image \( i \) belongs to class \( k \)

- **Learning objective:**
  \[
  E(\Phi, w_1..K, b_1..K) = \sum_{k=1}^{K} \left\{ \frac{1}{2} \|w_k\|^2 + \frac{\lambda}{N} \sum_{i=1}^{N} \ell \left[ y_{i,k} (w_k^T \Phi(x_i) + b_k) \right] \right\}
  \]

  \( \ell[.] \): hinge function

  tradeoff between large margin and misclassification over the \( K \) training classes
Linearization of a multiple kernel combiner

• “Sidestepping” the kernel trick [Vedaldi and Zisserman, 2010]: for the family of additive kernels there exists an explicit feature map \( \hat{\psi} \)

\[
x = \begin{bmatrix}
\text{GIST} \\
\text{self-similarity descriptor} \\
\text{oriented gradients} \\
\text{SIFT}
\end{bmatrix}
\]

\[
\hat{\psi} : \mathbb{R}^D \rightarrow \mathbb{R}^{D(2r+1)}
\]

such that \( K(x, x') \approx \langle \hat{\psi}(x), \hat{\psi}(x') \rangle \) for small \( r \) (we use \( r = 1 \)).

• Each basis classifier \( \phi_c \) can be approximated as a binarized linear projection in the 3D-dimensional space:

\[
\phi_c(z) = 1[a_c^T z]
\]
The final learning objective

\[
E(a_{1..C}, w_{1..K}, b_{1..K}) = \sum_{k=1}^{K} \left\{ \frac{1}{2} \|w_k\|^2 + \frac{\lambda}{N} \sum_{i=1}^{N} \ell \left[ y_{i,k}(\sum_{c=1}^{C} w_{k,c} [a_c^T z_i] + b_k) \right] \right\}
\]

Optimization via alternation:

- **learn linear classifiers** \((w_k, b_k)\) (while keeping \(a_{1..C}\) fixed):
  traditional linear SVM learning

- **learn PiCoDes projections** \(a_{1..C}\) (while keeping \((w_k, b_k)\) fixed):
  we optimize a convex upper bound of the objective that can be formulated as a linear program
Implementation details

We use spatial pyramid histograms of 4 low-level features, yielding a total of 13 histograms.

We choose the mapping \( \psi(x) \) that approximates the histogram intersection kernel.

\[ D = 17K \] but we reduce the dimensionality of the learning space to \( d = 6K \) via PCA.

\[ \hat{\psi} : \mathbb{R}^D \rightarrow \mathbb{R}^{D(2r+1)} \]

\[ z = \hat{\psi}(x) \]
Prior work on compact image codes

[Andoni and Indyk, 2006]; [Salakhutdinov and Hinton, 2009];
[Torralba et al., 2008]; [Ranzato et al., 2007]; [Weiss et al., 2008];
[Jegou et al., 2010]; [Perronnin and Sanchez, 2011], [Gong and
Lazebnik, 2011] ...

• **Given:**
  
  image descriptor \( x \in \mathcal{R}^D \) (e.g., GIST)

• **Learn:**
  
  compact code \( y \in \{0, 1\}^{D'} \)

  such that \( y_i \) is “near” \( y_j \) \( \iff \) \( x_i \) is “near” \( x_j \)
Experimental setup

Offline training set (PiCoDes learning):

- ImageNet (a subset)
  » 2625 classes
  » 30 examples / class

Evaluation database (PiCoDes testing):

- Caltech 256
  » 256 classes
  » 10 training ex / class
  » 25 test examples / class

- ImageNet ILSVRC 2010
  » 1000 classes
  » varying # tr ex / class
  » 150 test examples / class

no classes in common

Experiments:

- Multiclass recognition and object-class search
  » we use a linear SVM as classification model
Experiment 1: multiclass recognition on Caltech256

10 training examples/class

Linear SVM on x

LP−β (23 Kbytes/image)
Computational cost comparison

Training time

- **LPbeta**: 23 hours
- Linear SVM with PiCoDes: 30 seconds

Testing time

- **LPbeta**: 40 ms
- Linear SVM with PiCoDes: 30 ms
Experiment 2: novel-class search in ImageNet ILSVRC2010 (150K images, 1000 classes)

- for each query class the database contains 150 true positives, ~150K distractors
- random performance is about 0.1%
- training + search takes less than 1 second
Visualization of PiCoDes

- What kind of information is encoded in PiCoDes?

Images with large negative PiCoDes values (before binarization)

Images with large positive PiCoDes values (before binarization)
Conclusions

• PiCoDes:
  - binary features explicitly optimized for linear classification
  - can be trained for any desired descriptor size

• Even when reduced to about 200 bytes/image, recognition accuracy is similar to the best known MKL at a tiny fraction of the cost

• Future work:
  - features optimized for sparse/conjunctive classifiers
  - descriptors for subwindow recognition ([Li et al. NIPS10])

• *Software for (fast!) extraction of PiCoDes is available at:*