Efficient Match Kernels between Sets of Features for Visual Recognition
L. Bo, C. Sminchisescu

Chris Russell

April 20, 2010
Textual Bag of Words

- Stem words.
Textual Bag of Words

- Stem words.
- Build distribution of words in document
Textual Bag of Words

- Stem words.
- Build distribution of words in document
Textual Bag of Words

- Stem words.
- Build distribution of words in document

- Match distributions.
Visual Bag of Words

We want to do the same for images.

- Cluster features.
Visual Bag of Words

We want to do the same for images.

- Cluster features.
- Build distribution of binned features in image.
Visual Bag of Words

We want to do the same for images.

- Cluster features.
- Build distribution of binned features in image.
Visual Bag of Words

We want to do the same for images.

- Cluster features.
- Build distribution of binned features in image.

- Match distributions.
Match Kernels

This doesn’t work perfectly
Unlike document matching, information is lost by clustering

\[ K_B(X, Y) = \sum_{x \in X} \sum_{y \in Y} \delta(x, y) |X||Y| \quad (1) \]

where \( \delta(x, y) = \begin{cases} 1 & \text{if } x, y \in v_i \\ 0 & \text{otherwise} \end{cases} \) \quad (2)

What we actually want to solve is:

\[ K_S(X, Y) = \sum_{x \in X} \sum_{y \in Y} k(x, y) |X||Y| \quad (3) \]

where \( k \) is a local kernel.
Match Kernels

This doesn’t work perfectly

Unlike document matching, information is lost by clustering

- Linear matching of histograms can be expressed as a kernel
Match Kernels

This doesn’t work perfectly
Unlike document matching, information is lost by clustering

- Linear matching of histograms can be expressed as a kernel
- We can write the kernel as

\[
K_B(X, Y) = \frac{\sum_{x \in X} \sum_{y \in Y} \delta(x, y)}{|X||Y|}
\]

where

\[
\delta(x, y) = \begin{cases} 
1 & \text{where } x, y \in v_i \\
0 & \text{otherwise.}
\end{cases}
\]
Match Kernels

This doesn’t work perfectly

Unlike document matching, information is lost by clustering

- Linear matching of histograms can be expressed as a kernel
- We can write the kernel as

\[
K_B(X, Y) = \frac{\sum_{x \in X} \sum_{y \in Y} \delta(x, y)}{|X||Y|}
\]  

(1)

where

\[
\delta(x, y) = \begin{cases} 
1 & \text{where } x, y \in v_i \\
0 & \text{otherwise.} 
\end{cases}
\]  

(2)

- What we actually want to solve is:

\[
K_S(X, Y) = \frac{\sum_{x \in X} \sum_{y \in Y} k(x, y)}{|X||Y|}
\]  

(3)

where \(k\) is a local kernel.
Approximating $k$ with $\delta$

- $\delta(x, y)$ is a good approximation of $k(x, y)$ in two important case.
Approximating $k$ with $\delta$

- $\delta(x, y)$ is a good approximation of $k(x, y)$ in two important cases.

- If $x \approx y$ then $k(x, y) \approx 1 = \delta(x, y)$.
Approximating $k$ with $\delta$

- $\delta(x, y)$ is a good approximation of $k(x, y)$ in two important cases.
- If $x \approx y$ then $k(x, y) \approx 1 = \delta(x, y)$.
- If $x \not\approx z$ then $k(x, z) \approx 0 = \delta(x, z)$.
Approximating $k$ with $\delta$ (Failure cases)

\[ k(x, u) \not= 1 = \delta(x, u). \]
Approximating $k$ with $\delta$ (Failure cases)

- $k(x, u) \neq 1 = \delta(x, u)$.
- $k(u, v) \neq 0 = \delta(u, v)$. 
Approximating $k$ with $\delta$ (Failure cases)

- $k(x, u) \not\approx 1 = \delta(x, u)$.
- $k(u, v) \not\approx 0 = \delta(u, v)$.
- These failure cases occur more frequently in high dimensional spaces.
Match Kernels

- Define $k$ as a dot product over a finite dimensional space $\Phi$
Match Kernels

- Define $k$ as a dot product over a finite dimensional space $\Phi$

$$k(x, y) = \phi(x) \cdot \phi(y) \quad (4)$$

hence

$$K_S(X, Y) = \overline{\phi(x)} \cdot \overline{\phi(y)} \quad (5)$$

where $\overline{\phi(x)} = \sum_{x \in X} \phi(x) |X|$ is an EMK
Match Kernels

- Define $k$ as a dot product over a finite dimensional space $\Phi$

\[ k(x, y) = \phi(x) \cdot \phi(y) \] (4)

hence

\[ K_S(X, Y) = \bar{\phi}(x) \cdot \bar{\phi}(y) \] (5)

where

\[ \bar{\phi}(x) = \frac{\sum_{x \in X} \phi(x)}{|X|} \] (6)
Match Kernels

- Define $k$ as a dot product over a finite dimensional space $\Phi$
  \[
  k(x, y) = \phi(x) \cdot \phi(y)
  \] (4)

  hence

- 
  \[
  K_S(X, Y) = \bar{\phi}(x) \cdot \bar{\phi}(y)
  \] (5)

  where
  \[
  \bar{\phi}(x) = \frac{\sum_{x \in X} \phi(x)}{|X|}
  \] (6)

- Such a $K_S$ is called an Efficient Match Kernel (EMK)
Match Kernels

- Define $k$ as a dot product over a finite dimensional space $\Phi$

\[
k(x, y) = \phi(x) \cdot \phi(y)
\] (4)

hence

\[
K_S(X, Y) = \bar{\phi}(x) \cdot \bar{\phi}(y)
\] (5)

where

\[
\bar{\phi}(x) = \frac{\sum_{x \in X} \phi(x)}{|X|}
\] (6)

- Such a $K_s$ is called an Efficient Match Kernel (EMK)

\[
\frac{1}{|X||Y|} \sum_x \sum_y \delta(x, y)
\] is an EMK
Low Dimensional Projections

Efficient Kernel Projections

Given $Z = [z_1 \ldots z_n]$ a set of basis vectors, and
$H = [\phi(z_1), \ldots, \phi(z_D)]$
Efficient Kernel Projections

1. Given $Z = [z_1 \ldots z_n]$ a set of basis vectors, and $H = [\phi(z_1), \ldots, \phi(z_D)]$
2. We approximate $\phi(x) \approx Hv_x$
Low Dimensional Projections

Efficient Kernel Projections

- Given $Z = [z_1 \ldots z_n]$ a set of basis vectors, and $H = [\phi(z_1), \ldots, \phi(z_D)]$
- We approximate $\phi(x) \approx Hv_x$
- the best approximation is $\text{arg min}_{v_x} \|\phi(x) - Hv_x\|^2$. 
Efficient Kernel Projections

- Given $Z = [z_1 \ldots z_n]$ a set of basis vectors, and $H = [\phi(z_1), \ldots, \phi(z_D)]$
- We approximate $\phi(x) \approx Hv_x$
- the best approximation is $\arg \min_{v_x} ||\phi(x) - Hv_x||^2$.
- The problem is convex and can also be solved analytically.
Can we learn a good $Z$?

- Kernel PCA?
Can we learn a good $Z$?

- Kernel PCA?
- Pick $F$ random features
Can we learn a good $Z$?

- Kernel PCA?
- Pick $F$ random features
- Pick $D$ largest eigenvectors.
Can we learn a good $Z$?

- Kernel PCA?
- Pick $F$ random features
- Pick $D$ largest eigenvectors.
- Training is $O(F^3)$
Can we learn a good $Z$?

- Kernel PCA?
- Pick $F$ random features
- Pick $D$ largest eigenvectors.
- Training is $O(F^3)$
- Evaluation is $O(FmD)$
Constrained kernel SVD

- Enforce sparsity
  Following KPCA, pick the best \( \bar{z} \)
  By choosing

\[
(\bar{z}, \bar{\beta}) = \arg \min_{\bar{z}, \bar{\beta}} \sum_{i \leq F} ||\alpha_i \cdot \phi(x_i) - \bar{\beta} \phi(\bar{z})||
\]

(7)
Constrained kernel SVD

- Enforce sparsity
  Following KPCA, pick the best $\bar{z}$
  By choosing

  $$(\bar{z}, \bar{\beta}) = \arg\min_{\bar{z}, \bar{\beta}} \sum_{i \leq F} ||\alpha_i \cdot \phi(x_i) - \bar{\beta}\phi(\bar{z})||$$

  (7)

- This is sub-optimal
  Instead perform Constrained Kernel SVD (CKSVD)

  $$\arg\min_{V, Z} R(V, Z) = \frac{\sum_{i \leq F} ||\alpha_i \cdot \phi(x_i) - H\nu_i||^2}{F}$$

  (8)
Efficient CKSVD

- Eliminate $V$ by enforcing $\frac{\partial R}{\partial v_i} = 0$
Efficient CKSVD

- Eliminate $V$ by enforcing $\frac{\partial R}{\partial v_i} = 0$

\[
\arg \min_Z R^*(Z) = -\frac{1}{F} \sum_{i \leq F} k_Z(x_i)^T K_{ZZ}^{-1} k_Z(x_i)
\tag{9}
\]

where $k_Z$ is a $D$ dimensional vector such that $\{k_Z\}_i = k(x, Z)$

$K_{ZZ}$ is a $D \times D$ matrix where $\{K_{ZZ}\}_{ij} = k(z_i, z_j)$. 

Efficient CKSVD

- Eliminate $V$ by enforcing $\frac{\partial R}{\partial v_i} = 0$

$$\arg \min_Z R^*(Z) = -\frac{1}{F} \sum_{i \leq F} k_Z(x_i)^T K_{ZZ}^{-1} k_Z(x_i)$$ \hspace{1cm} (9)

where $k_Z$ is a $D$ dimensional vector such that $\{k_z\}_i = k(x, Z)$

$K_{ZZ}$ is a $D \times D$ matrix where $\{K_{ZZ}\}_{ij} = k(z_i, z_j)$.

- The resulting problem can be solved with SGD at a cost of $O(D^3)$ per iteration.
No guarantees

- Technique minimises kernel reconstruction error
No guarantees

- Technique minimises kernel reconstruction error
- Not guaranteed to preserve discriminative projections
No guarantees

- Technique minimises kernel reconstruction error
- Not guaranteed to preserve discriminative projections
- Similar to PCA vs Fishers linear discriminate
Comparison with Random Fourier Approximation

Reconstruction Error
Left: Learnt Low dimensional Approximation with 20 features
Centre: Training Error under SGD
Right: Fourier Approximation 200 random features
Complexity of kernel computation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>$O(n^2 m^2 d)$</td>
<td>$O(n^2 m^3 d)$</td>
<td>$O(n^2 m \log(T)d)$</td>
<td>$O(nmDd + nD^2)$</td>
<td>$O(nmDd)$</td>
</tr>
<tr>
<td>Test</td>
<td>$O(nm^2 d)$</td>
<td>$O(nm^3 d)$</td>
<td>$O(nm \log(T)d)$</td>
<td>$O(mDd + D^2)$</td>
<td>$O(mDd)$</td>
</tr>
</tbody>
</table>

$n$ number of images

$m$ number of words

$d$ dimensionality of feature vector

$D$ dimensionality of $\Phi$
Results — Scene 15

![Graphs showing accuracy vs. dimensionality and training set size for different kernels: BOW-Linear, BOW-Gaussian, EMK-CKSVD, EMK-Fourier.](image)
### Results — Caltech 101

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>15 training</th>
<th>30 training</th>
<th>Algorithms</th>
<th>15 training</th>
<th>30 training</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMK [5, 6]</td>
<td>50.0±0.9</td>
<td>58.2</td>
<td>kCNN [30]</td>
<td>59.2</td>
<td>67.4</td>
</tr>
<tr>
<td>HMAX [19]</td>
<td>51.0</td>
<td>56.0</td>
<td>LDF [4]</td>
<td>60.3</td>
<td>N/A</td>
</tr>
<tr>
<td>ML+PMK [9]</td>
<td>52.2</td>
<td>62.1</td>
<td>ML+CORR [9]</td>
<td>61.0</td>
<td>69.6</td>
</tr>
<tr>
<td>KC [28]</td>
<td>N/A</td>
<td>64.0</td>
<td>NBNN [1]</td>
<td>65.0±1.1</td>
<td>73.0</td>
</tr>
<tr>
<td>SPM [14]</td>
<td>56.4</td>
<td>64.4±0.5</td>
<td>EMK-Fourier</td>
<td>60.2±0.8</td>
<td>70.1±0.8</td>
</tr>
<tr>
<td>SVM-KNN [31]</td>
<td>59.1±0.5</td>
<td>66.2±0.8</td>
<td>EMK-CKSVD</td>
<td>60.5±0.9</td>
<td>70.3±0.8</td>
</tr>
</tbody>
</table>
Table 4: Accuracy on Caltech-256. The results are averaged over five random training/testing splits. The dimensionality of the feature maps and the vocabulary size are both set to 1000 (for fair comparisons). We use 2 pyramid levels.