The Problem

- For any general recognition task, there is usually a database of labeled images.
- When a novel image is seen, a distance is computed between this image and every image in the database.
The Problem: an illustration

The distance function can be anything! Can be non-metric, bizarre, etc.
Each query requires \( n \) distance calculations for a database of size \( n \).
What if the distance function is very complicated and expensive computationally?
The Solution: BoostMap

- BoostMap is a method that can reduce the number of expensive distance calculations down to some $d << n$
- It works for ANY distance function

Formalities

- Let $X$ be a set of objects, and $D_X(x_1, x_2)$ be a distance measure between objects of this set.
- Let $(q, x_1, x_2)$ be a triplet of objects from the set
- Define the Proximity Function $P_X(q, x_1, x_2)$

$$P_X(q, x_1, x_2) = \begin{cases} 
1 & \text{if } D_X(q, x_1) < D_X(q, x_2) \\
0 & \text{if } D_X(q, x_1) = D_X(q, x_2) \\
-1 & \text{if } D_X(q, x_1) > D_X(q, x_2) 
\end{cases}$$
Formalities

- Suppose we had an embedding $F: X \rightarrow \mathbb{R}^d$
- Let $P_R$ be proximity function of $F(X)$ that uses some metric distance $D_R$ (e.g. $L_1$, $L_2$, etc)

$$P_R (q, x_1, x_2) = \begin{cases} 
1 & \text{if } D_R (q, x_1) < D_R (q, x_2) \\
0 & \text{if } D_R (q, x_1) = D_R (q, x_2) \\
-1 & \text{if } D_R (q, x_1) > D_R (q, x_2) 
\end{cases}$$

Formalities

- Define a Proximity Classifier $\bar{F}(q, x_1, x_2)$

$$\bar{F}(q, x_1, x_2) = P_R^d (F(q), F(x_1), F(x_2))$$

- We want $\bar{F}$ to output the same thing as $P_X$
Computing Error

For a single triple \((q,x_1,x_2)\)

\[
G(F, q, x_1, x_2) = \frac{|P_X(q, x_1, x_2) - F(q, x_1, x_2)|}{2}
\]

For all your data

\[
G(F) = \frac{\sum_{(q,x_1,x_2) \in X^3} G(F, q, x_1, x_2)}{|X|^3}.
\]

How do we get the embedding \(F\)?

Let’s think about simpler embeddings \(F: X \rightarrow R\)

Generate many random simple embeddings and throw them into AdaBoost

Our final embedding will be a linear combination of the simple embeddings
1D Embeddings

- Use a reference object \( r \)

Classifies 46 out of 60 triplets correct. Incorrect: \((b, a, c); (c, b, d); (d, b, r)\)

1D Embeddings

- Use “pivot points”
Boost 1D embedding

- How many people are not familiar with boosting?
- Use training data (which can be generated by using the original distance function $D_x$)
- AdaBoost outputs a set of $d$ 1D embeddings, and a weight for each.

Final BoostMap Embedding

- Weighted L1 distance that combines the chosen 1D embeddings and their weights.
- Suppose we chose $d$ embeddings. To compute the embedded distance between $X_u$ and $X_v$:

$$D_{R^d}(u_1, ..., u_d), (v_1, ..., v_d)) = \sum_{j=1}^{d} (\alpha_j |u_j - v_j|)$$
What do we end up with?

- An embedding $F: X \rightarrow \mathbb{R}^d$ which uses up to $2d$ reference objects.
- A weighted L1 metric in this $\mathbb{R}^d$ space.
- We know that the embedding in some sense preserves the proximity.

At Run-time

- Suppose we want to compare object $Q$ (query) to objects $X_1, X_2, \ldots, X_n$ in the DB.
- Need to compute $d$ embeddings of $Q$: $O(d)$ calls to $D_x$.
- Compute weighted L1 distance between $Q$ and $X_1, X_2, \ldots, X_n$ – much cheaper than computing $D_x n$ times.
Does it work?

- Hand experiment

- Original distance measure: Chamfer distance (takes 260s to query)

---

### Does it work?

- Hand experiment

<table>
<thead>
<tr>
<th>ENN retrieval accuracy and efficiency for hand database</th>
<th>BoostMap</th>
<th>FastMap</th>
<th>Exact $D_X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENN-accuracy</td>
<td>95%</td>
<td>100%</td>
<td>95%</td>
</tr>
<tr>
<td>Best $d$</td>
<td>256</td>
<td>256</td>
<td>13</td>
</tr>
<tr>
<td>Best $p$</td>
<td>406</td>
<td>3850</td>
<td>3838</td>
</tr>
<tr>
<td>$D_X$ # per query</td>
<td>823</td>
<td>4267</td>
<td>3864</td>
</tr>
<tr>
<td>seconds per query</td>
<td>2.3</td>
<td>10.6</td>
<td>9.4</td>
</tr>
</tbody>
</table>
Does it work?

- Shape contexts

<table>
<thead>
<tr>
<th>Method</th>
<th>Distances per query object</th>
<th>Speed-up factor</th>
<th>Seconds per query object</th>
<th>Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>brute force</td>
<td>20,000</td>
<td>1</td>
<td>1232</td>
<td>0.63%</td>
</tr>
<tr>
<td>Vp-trees [24]</td>
<td>8594</td>
<td>2.3</td>
<td>572</td>
<td>0.66%</td>
</tr>
<tr>
<td>CNN [10]</td>
<td>1060</td>
<td>18.9</td>
<td>70.6</td>
<td>2.40%</td>
</tr>
<tr>
<td>Zhang [25]</td>
<td>50</td>
<td>400</td>
<td>3.3</td>
<td>2.55%</td>
</tr>
<tr>
<td>BoostMap</td>
<td>800</td>
<td>25</td>
<td>53.3</td>
<td>0.74%</td>
</tr>
<tr>
<td>BoostMap-C</td>
<td>800</td>
<td>25</td>
<td>53.3</td>
<td>0.72%</td>
</tr>
<tr>
<td>Cascade</td>
<td>149</td>
<td>134</td>
<td>9.9</td>
<td>0.75%</td>
</tr>
<tr>
<td>Cascade-C</td>
<td>92.5</td>
<td>216</td>
<td>6.2</td>
<td>0.74%</td>
</tr>
</tbody>
</table>

Questions?