**GRAPH REGULARISED HASHING**

**INTRODUCTION**

- Problem: Constant time nearest-neighbour search in large datasets.
- Hashing-based approximate nearest neighbour (NN) search:
  - Index image/documents into the buckets of hashtable(s)
  - Encourage collisions between similar images/documents

**RESEARCH QUESTION**

- Locality sensitive hashing (LSH) [1] fractures the input feature space space with randomly placed hyperplanes.
- Can we do better by using supervision to adjust the hyperplanes?

**GRAPH REGULARISED HASHING (GRH)**

- We propose a two step iterative hashing model, Graph Regularised Hashing (GRH) [5]. GRH uses supervision in the form of an adjacency matrix that specifies whether or not data-points are related.
  - **Step A: Graph Regularisation**: the K-bit hashcode of a data-point is set to the average of the data-points of its nearest neighbours as specified by the adjacency graph:
    \[ L_m \leftarrow \text{sgn} \left( \alpha \text{SD}^{-1}L_{m-1} + (1-\alpha)L_0 \right) \]
    - S: Affinity (adjacency) matrix
    - D: Diagonal degree matrix
    - L: Binary bits at iteration m
    - \( \alpha \in \{0,1\} \): Linear interpolation parameter
  - Step A is a simple sparse-sparse matrix multiplication, and can be implemented very efficiently. Any existing hash function e.g. LSH [1] can be used to initialise the bits in \( L_0 \).
  - **Step B: Data-Space Partitioning**: the hashcodes produced in Step A are used as the labels to learn \( k \) binary classifiers. This is the out-of-sample extension step, allowing the encoding of data-points not seen before:
    \[ \text{for } k = 1 \ldots K : \min \left\| w_k \right\|^2 + C \sum_{i=1}^{N_{trd}} \xi_{ik} \]
    \[ \text{s.t. } L_{ik} (w_k^T x_i + t_k) \geq 1 - \xi_{ik} \text{ for } i = 1 \ldots N_{trd} \]
    - \( w_k \): Hyperplane \( k \)
    - \( t_k \): bias \( k \)
    - \( x_i \): data-point \( i \)
    - \( L_{ik} \): bit \( k \) of data-point \( i \)
    - \( \xi_{ik} \): slack variable \( k \)
  - Steps A-B are repeated for a set number of iterations (M) e.g. \( < 10 \).
  - The learnt hyperplanes \( w_k \) can then be used to encode unseen data-points (via a simple dot-product).

**STEP A: GRAPH REGULARISATION**

- Toy example: nodes are images with 3-bit LSH encoding. Arcs indicate nearest neighbour relationships. We show two images \((c,e)\) having their hashcodes updated in Step A:

**STEP B: DATA-SPACE PARTITIONING**

- Here we show a hyperplane being learnt using the first bit as (highlighted with bold box) as label. One hyperplane is learnt per bit.

**QUANTITATIVE RESULTS (CIFAR-10) (MORE DATASETS IN PAPER)**

- Mean average precision (mAP) image retrieval results using GIST features on CIFAR-10 (▲▲ is significant at \( p < 0.01 \)):

<table>
<thead>
<tr>
<th>CIFAR-10</th>
<th>16 bits</th>
<th>32 bits</th>
<th>48 bits</th>
<th>64 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITQ+CCA [2]</td>
<td>0.2015</td>
<td>0.2130</td>
<td>0.2208</td>
<td>0.2237</td>
</tr>
<tr>
<td>STH [3]</td>
<td>0.2352</td>
<td>0.2702</td>
<td>0.2118</td>
<td>0.2000</td>
</tr>
<tr>
<td>KSH [4]</td>
<td>0.2496</td>
<td>0.2785</td>
<td>0.2849</td>
<td>0.2905</td>
</tr>
<tr>
<td>GRH [5]</td>
<td>0.2991▲▲</td>
<td>0.3122▲▲</td>
<td>0.3252▲▲</td>
<td>0.3350▲▲</td>
</tr>
</tbody>
</table>

- Timings (seconds) averaged over 10 runs. GRH is 1) faster to train and 2) is faster to encode unseen data-points:

<table>
<thead>
<tr>
<th>Training</th>
<th>Testing</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRH [5]</td>
<td>8.01</td>
<td>0.03</td>
</tr>
<tr>
<td>KSH [4]</td>
<td>74.02</td>
<td>0.10</td>
</tr>
<tr>
<td>BRE [6]</td>
<td>227.84</td>
<td>0.37</td>
</tr>
</tbody>
</table>

**SUMMARY OF KEY FINDINGS**

- First both accurate and scalable supervised hashing model
- Future work will extend GRH to streaming data sources
- Code online: https://github.com/sjmoran/grh

**REFERENCES**

- [2] GIST features on CIFAR-10 (M = 10).