Graph Regularised Hashing

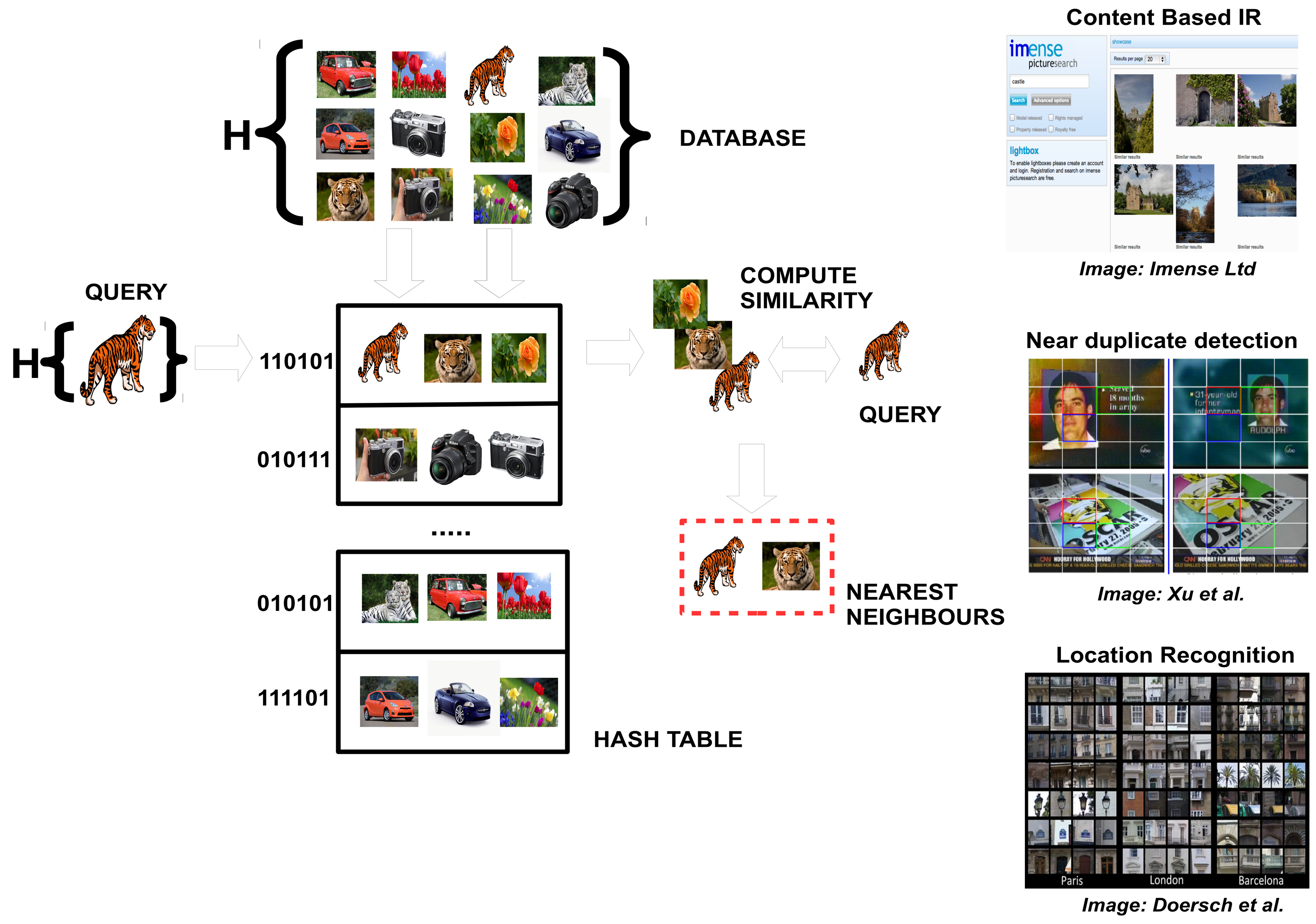
Sean Moran†, Victor Lavrenko
† sean.moran@ed.ac.uk

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?

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



- **Advantages:**
  - O(1) lookup per query rather than O(N) (brute-force)
  - Memory/storage saving due to compact binary codes

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:

$$\mathbf{L}_m \leftarrow \text{sgn}\left(\alpha\, \mathbf{S}\mathbf{D}^{-1}\mathbf{L}_{m-1} + (1-\alpha)\mathbf{L}_0\right)$$

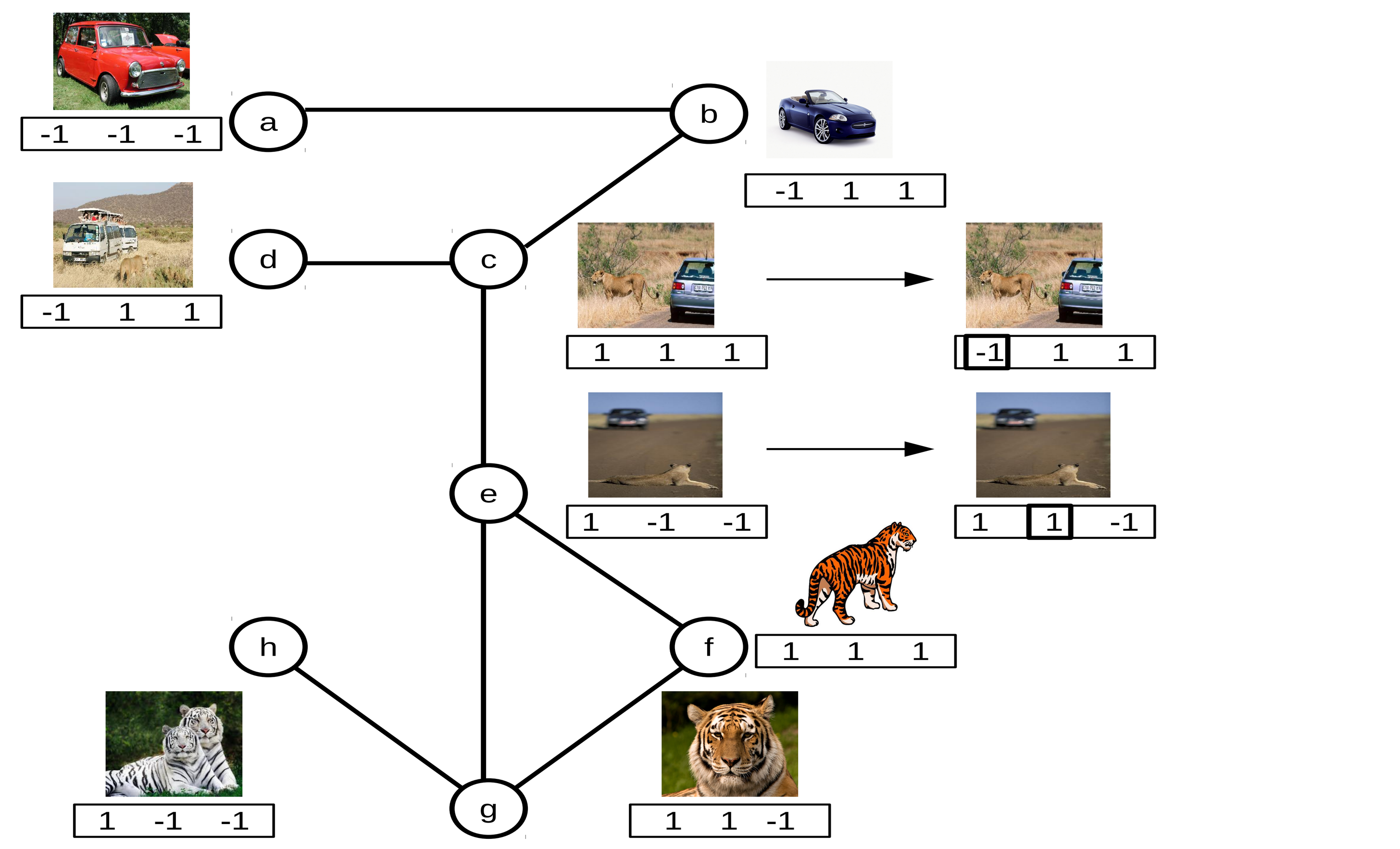
    - **S**: Affinity (adjacency) matrix
    - **D**: Diagonal degree matrix
    - **L**: Binary bits at iteration m
    - $\alpha \in \{0, 1\}$: Linear interpolation parameter
  - Step A is a simple sparse-sparse matrix multiplication, and can be implemented very efficiently. Any existing hash function e.g. LSH [1] can be used to initialise the bits in $\mathbf{L}_0$
  - **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:

$$\text{for } k = 1 \ldots K: \quad \min \ ||\mathbf{w}_k||^2 + C\sum_{i=1}^{N_{trd}} \xi_{ik}$$
$$\text{s.t.} \quad L_{ik}(\mathbf{w}_k{}^{\intercal}\mathbf{x}_i + t_k) \geq 1 - \xi_{ik} \qquad \text{for } i = 1 \ldots N_{trd}$$

    - $\mathbf{w}_k$: Hyperplane k $\quad t_k$: bias k
    - $\mathbf{x}_i$: data-point i $\quad L_{ik}$: bit k of data-point i
    - ξ_{ik}: slack variable $\quad K$: # bits $\quad N_{trd}$: # data-points
- Steps A-B are repeated for a set number of iterations (M) (e.g. < 10). The learnt hyperplanes $\mathbf{w}_k$ can then be used to encode unseen data-points (via a simple dot-product).

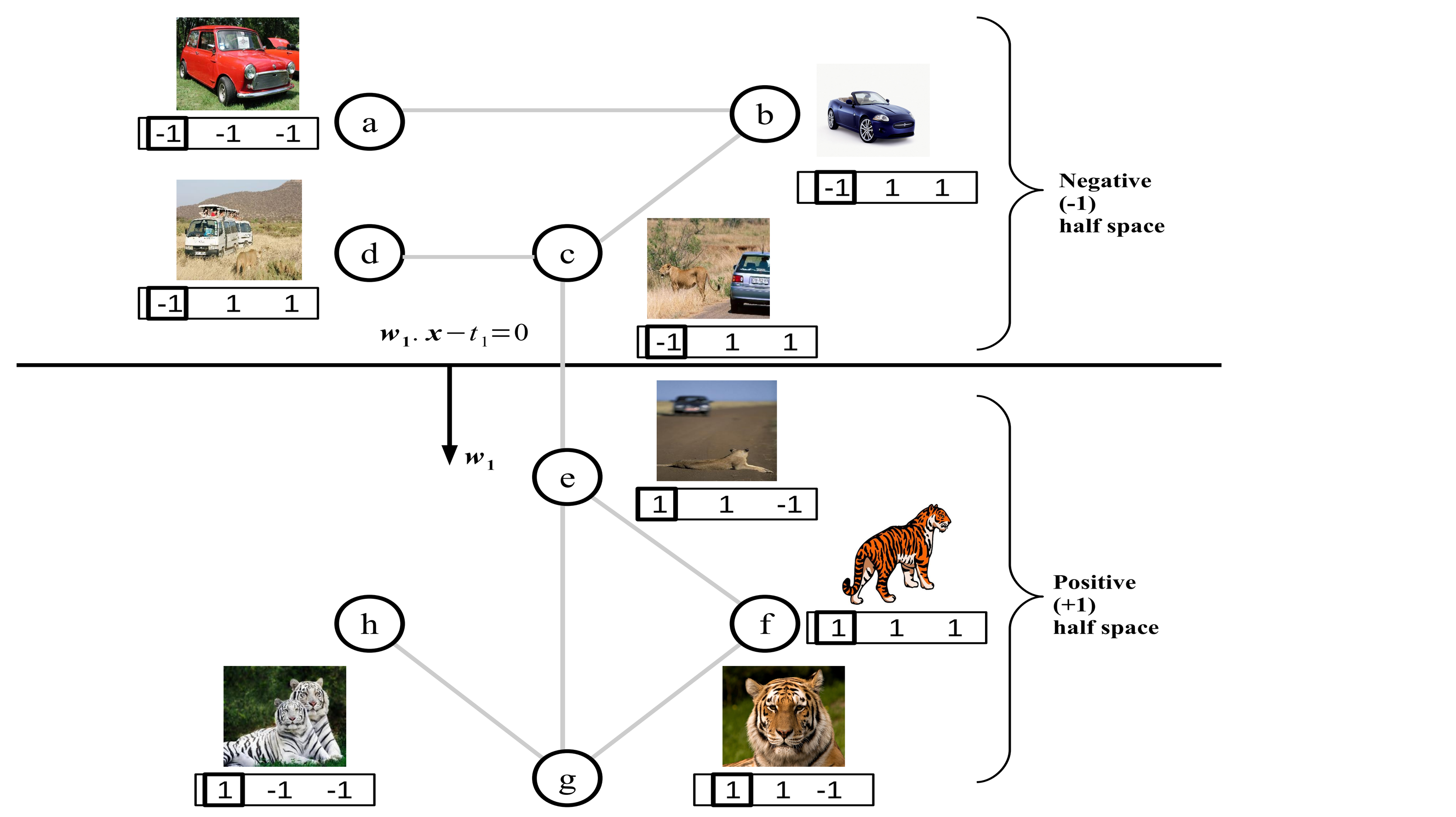
Step A: Graph Regularisation

- 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 $p < 0.01$):

CIFAR-10				
	16 bits	32 bits	48 bits	64 bits
ITQ+CCA [2]	0.2015	0.2130	0.2208	0.2237
STH [3]	0.2352	0.2072	0.2118	0.2000
KSH [4]	0.2496	0.2785	0.2849	0.2905
GRH [5]	**0.2991▲▲**	**0.3122▲▲**	**0.3252▲▲**	**0.3350▲▲**

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

	Training	Testing	Total
GRH [5]	8.01	0.03	8.04
KSH [4]	74.02	0.10	74.12
BRE [6]	227.84	0.37	228.21

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:

[1] P. Indyk, R. Motwani: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: STOC (1998).

[2] Y. Gong, S. Lazebnik: Iterative Quantisation. In: CVPR (2011). , [3] D. Zhang et al. Self-Taught Hashing. In: SIGIR (2010)., [4] W. Liu et al. Supervised Hashing with Kernels. In: CVPR (2012)., [5] S. Moran, V. Lavrenko. Graph Regularised Hashing. In: ECIR (2015).,

[6] B. Kulis, T. Darrell et al. Binary Reconstructive Embedding. In: NIPS (2009).