CVPR JUNE 19-24 2022 NEW ORLEANS LOUISIANA

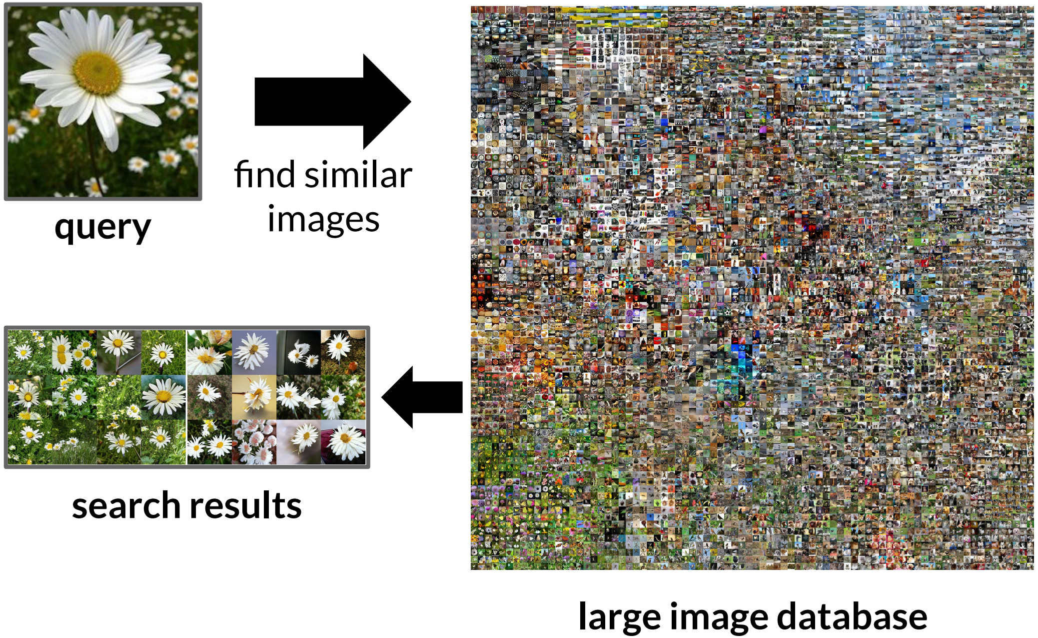
One Loss for Quantization: Deep Hashing with Discrete Wasserstein Distributional Matching

Khoa D. Doan [khoadoan.me], Peng Yang, and Ping Li
Cognitive Computing Lab, **Baidu Research, USA**

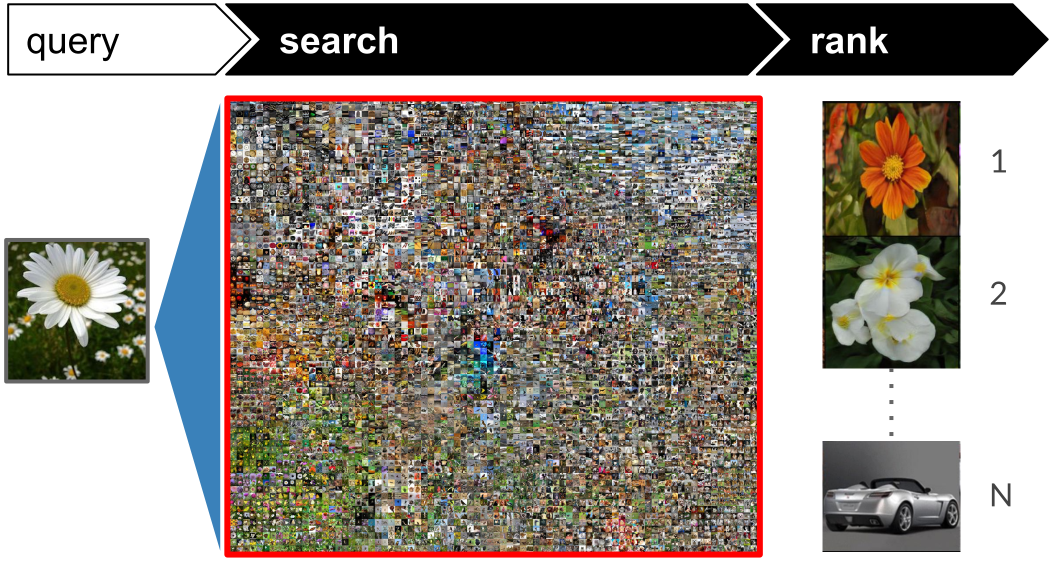
Similarity Search

Problem: Given a dataset of N items $X = \{x_1, x_2, \ldots, x_N\}$ and a query q, we aim to find l items $R = \{x_1, x_2, \ldots, x_l\}$ such that, for a similarity function $\mathbf{sim}$, we have:

$$\mathbf{sim}(q, x_i) \geq \mathbf{sim}(q, x_j) \quad \forall x_i \in R, \forall x_j \in X \backslash R$$



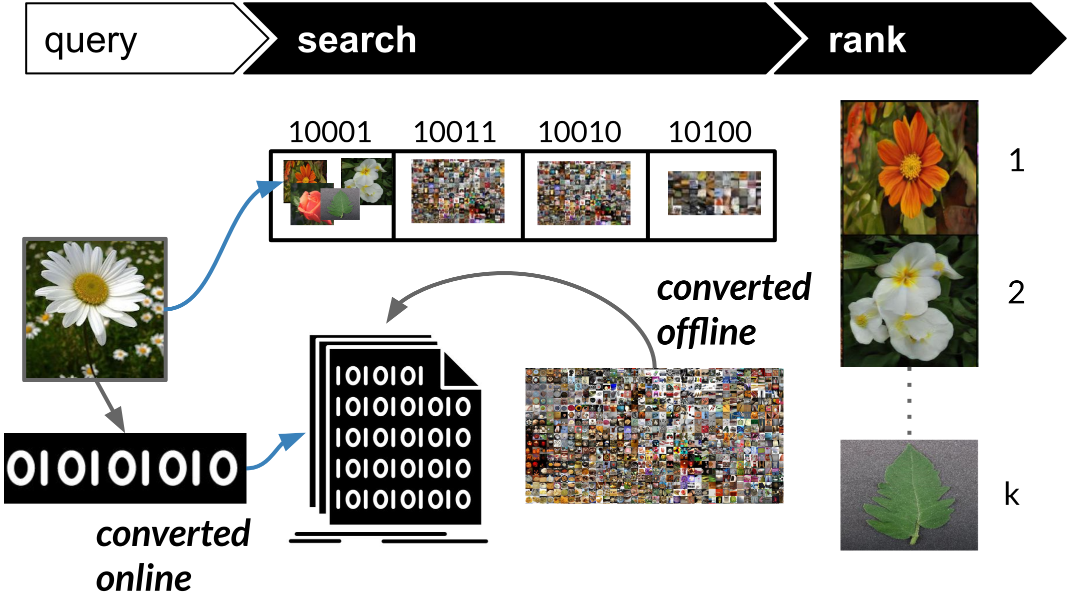
Linear Search



Exhaustive search

- ▷ Infeasible in large database of millions or billions of items.
- ▷ Wasteful of computation
  - ○ only a small subset is relevant
  - ○ real-time ranking is impossible

Approximate Nearest Neighbor (ANN)



Approximate Search (Hashing)

- Transforms images into binary vectors expressing their similarity.
- Search via table look-up
- Linear Search in Discrete space:
  - Memory efficient: 4MB for 1M items
  - Compute efficient: 2 instructions per distance computation

Hash-function Learning

- Learn a hash function

$$F : \mathcal{R}^n \longrightarrow \{0, 1\}^m$$

discrete function

$$f : \mathcal{R}^n \longrightarrow [0, 1]^m$$

continuous relaxation

$$F(x) = f(x) > 0.5$$

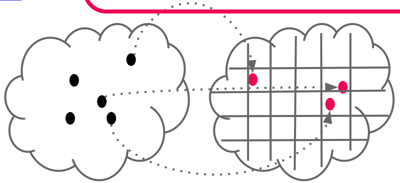
discretization

- Overall objective function of hashing methods

this work

$$\arg\min_f E_{x \sim D_x} L(x, f(x)) + E_{x \sim D_x} \sum_k \lambda_i \times H_k(f(x))$$

locality-preserving loss
preserves the semantics of $\mathbf{sim}$ in discrete space



hashing regularizer
minimizes gap between continuous and discrete optimizations.

Hash-function Learning

- Learn a hash function

$$F : \mathcal{R}^n \longrightarrow \{0, 1\}^m$$

discrete function

$$f : \mathcal{R}^n \longrightarrow [0, 1]^m$$

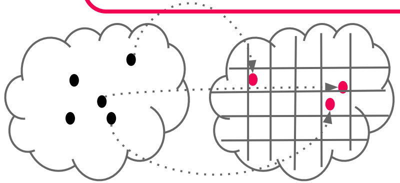
continuous relaxation

$$F(x) = f(x) > 0.5$$

discretization

- Overall objective function of hashing methods

$$\arg\min_f E_{x\sim D_x} L(x, f(x)) + E_{x\sim D_x} \sum_k \lambda_i \times H_k(f(x))$$



Existing Objectives are Complex

$\min_f$ [locality preserving loss]

Bit Balance

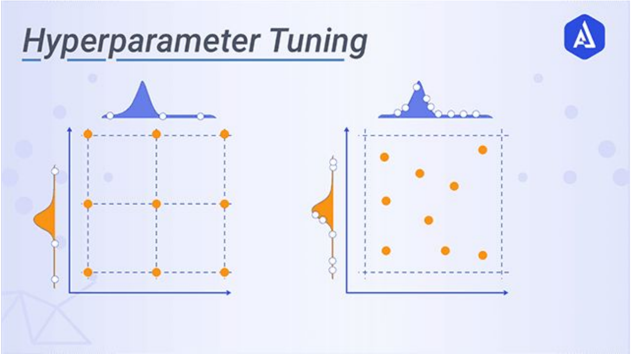
$$+\left|W^T W - I\right|_2 + \sum_{k=1}^{m} \bar{b}_k \log \bar{b}_k + (1 - \bar{b}_k) \log(1 - \bar{b}_k)$$

Bit Uncorrelation

$$+\sum_x \sum_{k=1}^{m} - f(x) \log(f(x)) - (1 - f(x)) \log(1 - f(x))$$

Low Quantization Error

Complex objective increases **training complexity** (i.e., hyperparameter tuning)



[Source: Online]

Existing Objectives are Complex

$\min_f$ [locality preserving loss]

Bit Balance

$$+\left|W^TW - I\right|_2 + \sum_{k=1}^{m} \bar{b}_k \log\bar{b}_k + (1-\bar{b}_k)\log(1-\bar{b}_k)$$

Bit Uncorrelation

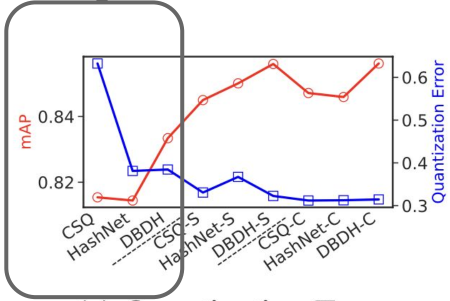
$$+\sum_x \sum_{k=1}^{m} -f(x)\log(f(x)) - (1-f(x))\log(1-f(x))$$

Low Quantization Error

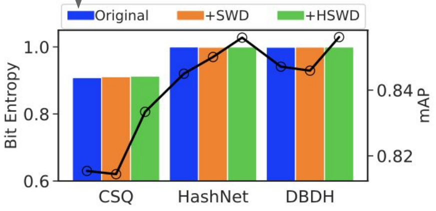
existing optimization

Complex objective increases **training complexity** (i.e., hyperparameter tuning)

Complex objective results in **sub-optimal quantization**



(a) Quantization Error



(b) Bit Entropy

[Doan et al. 2022]

Existing Objectives are Complex

$$\min_f \text{ [locality preserving loss]}$$

Bit Balance

$$+\left|W^TW - I\right|_2 + \sum_{k=1}^{m} \bar{b}_k \log\bar{b}_k + (1-\bar{b}_k)\log(1-\bar{b}_k)$$

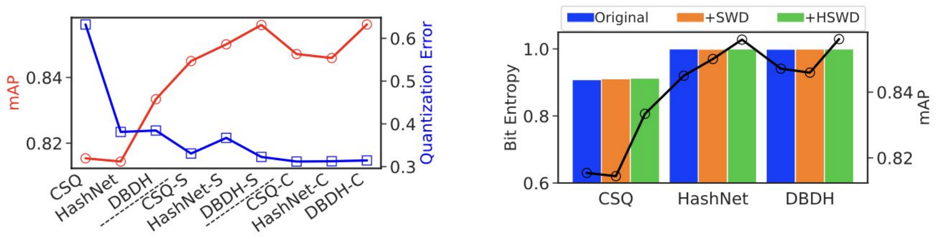
Bit Uncorrelation

$$+\sum_x \sum_{k=1}^{m} -f(x)\log(f(x)) - (1-f(x))\log(1-f(x))$$

Low Quantization Error

Complex objective increases **training complexity** (i.e., hyperparameter tuning)

Complex objective results in **sub-optimal quantization**



(a) Quantization Error (b) Bit Entropy

[Doan et al. 2022]

Existing Objectives are Complex

$$\min_f \text{ [locality preserving loss]}$$

Bit Balance

$$+\left|W^TW - I\right|_2 + \sum_{k=1}^{m} \bar{b}_k \log\bar{b}_k + (1-\bar{b}_k)\log(1-\bar{b}_k)$$

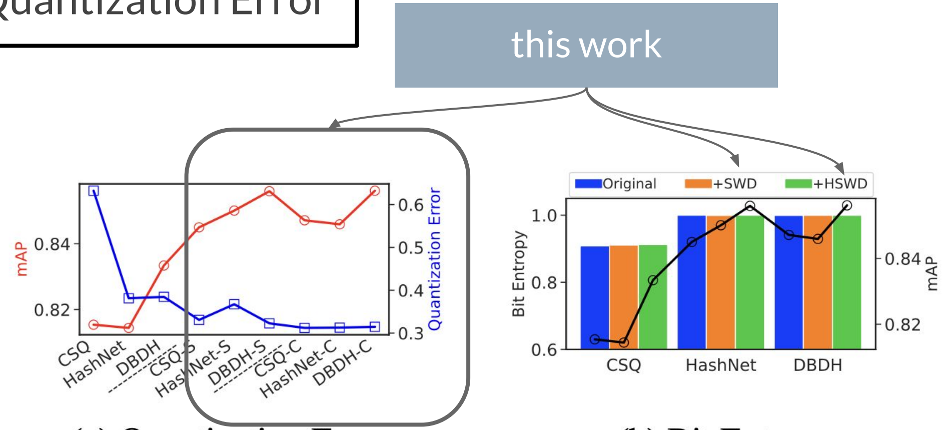
Bit Uncorrelation

$$+\sum_x \sum_{k=1}^{m} -f(x)\log(f(x)) - (1-f(x))\log(1-f(x))$$

Low Quantization Error

Complex objective increases **training complexity** (i.e., hyperparameter tuning)

Complex objective results in **sub-optimal quantization**



(a) Quantization Error (b) Bit Entropy

[Doan et al. 2022]

Single-shot Quantization Loss

Our approach: single divergence loss

$$\arg\min_f d(q \,||\, q^\star) \qquad f(x) \sim q$$

$q^\star$: fixed distribution

One Single Quantization Loss

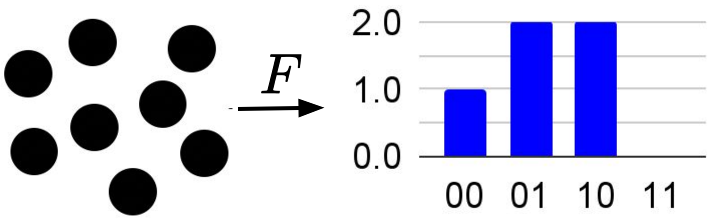
Disadvantages: challenging to optimize

$\min_f$ [locality preserving loss]

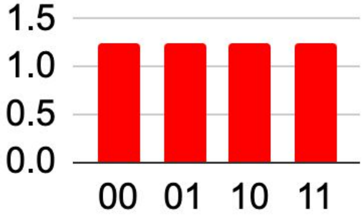
$$+ d(q \,||\, q^\star)$$

can be used to improve performance of any existing Deep Supervised Hashing

Task: learn 2-bit hash function



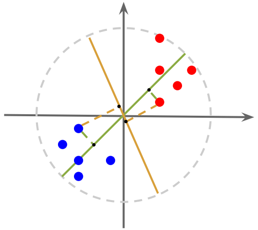
learned distribution q

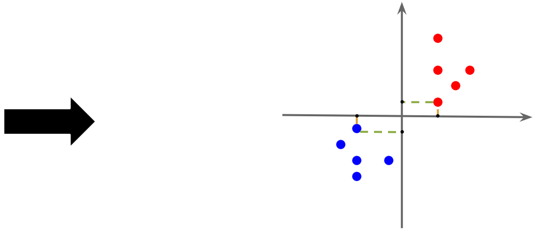


optimal distribution $q^\star$
(with maximum entropy)

$$q^\star : b_i \sim \text{bernoulli}(0.5)$$

Choosing the "Right" Divergence $\mathcal{D}(q(b) || q^{\star}(z))$





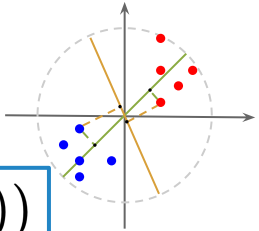
Sliced Wasserstein Distance
- Lower sample complexity
- No minimax
- Several directions are discriminative

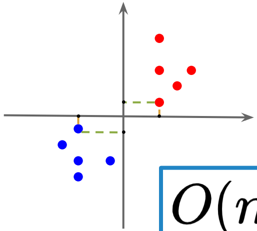
Hash-Sliced Wasserstein Distance
- Lower sample complexity
- No minimax
- Small number of discriminative projections

Wasserstein Distance
- Non-trivial to estimate
- High sample complexity
- Possibly minimax optimization (dual domain)

Other divergences (e.g. KL, JSD, etc...)
- Do not work with non-overlapping supports
- High sample complexity
- Minimax optimization

Choosing the "Right" Divergence $\mathcal{D}(q(b)||\, q^{\star}(z))$





$O(LN\log(Nd))$

Sliced Wasserstein Distance

- Lower sample complexity
- No minimax
- Several directions are discriminative

$O(mN\log(Nd)), m \ll L$

Proposed Hash-Sliced Wasserstein Distance

- Lower sample complexity
- No minimax
- Small number of discriminative projections

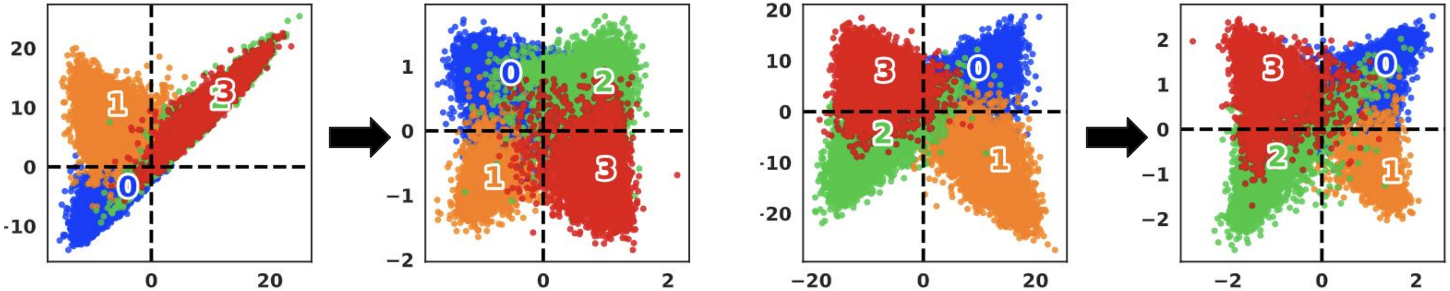
$$\mathcal{D}(h(X), B) \approx \left(\frac{1}{m}\sum_{l=1}^{m}[\mathcal{W}(h(X)_{l,:}, B_{l,:})]^2\right)^{1/2}$$

no projection: averaging along each hashing dimension

$$\mathcal{D}(h(X), B) \approx \left(\frac{1}{L}\sum_{l=1}^{L}\mathcal{W}(\omega_l^T h(X), \omega_l^T B)\right)^{1/2}$$

projection into 1-D space

Single-shot Quantization



Original HashNet (mAP 0.7208)	**HashNet/Single-Loss** (mAP 0.8500)	Original CSQ (mAP 0.8280)	**CSQ/Single-Loss** (mAP 0.8521)

Figure. Learn 2-bit hash function on CIFAR10's data from 4 classes

Table. Averaged running time per epoch across different supervised hashing methods (in seconds).

Dataset	Original	SWD	HSWD
CIFAR-10	19.4	24.2	17.1/**40%**
NUS-WIDE	58.3	71.2	50.1/**41%**
COCO	55.6	68.1	49.5/**37%**

More computationally efficient even before intensive model selection

Performance Evaluation (Precision@1000)

Retrieve k items Precision@k = number of / k **Blue: improvement over original methods**

-S: Sliced Wasserstein Estimate | -C: Proposed Wasserstein Estimate

Method	CIFAR-10		NUS-WIDE	
	16 bits	32 bits	16 bits	32 bits
DSDH	0.8252	0.8406	0.8117	0.8294
DSDH-S	0.8526/**3.3%**	0.8543/**1.6%**	0.8162/**0.6%**	0.8312/**0.2%**
DSDH-C	0.8645/**4.8%**	0.8739/**4.0%**	0.8195/**1.0%**	0.8391/**1.2%**
HashNet	0.6193	0.8613	0.7581	0.8158
HashNet-S	0.8470/**36.8%**	0.8755/**1.7%**	0.7743/**2.1%**	0.8199/**0.5%**
HashNet-C	0.7698/**24.3%**	0.8715/**1.2%**	0.7456/*-1.7%*	0.8078/*-1.0%*
GreedyHash	0.8561	0.8616	0.7601	0.8009
GreedyHash-S	0.8583/**0.3%**	0.8656/**0.5%**	0.7657/**0.7%**	0.7973/*-0.5%*
GreedyHash-C	0.8517/*-0.5%*	0.8700/**1.0%**	0.7630/**0.4%**	0.7931/*-1.0%*
DCH	0.8621	0.8568	0.7843	0.7898
DCH-S	0.8622/*0.0%*	0.8761/**2.3%**	0.7846/*0.0%*	0.7923/**0.3%**
DCH-C	0.8654/**0.4%**	0.8635/**0.8%**	0.7893/**0.6%**	0.7914/**0.2%**
CSQ	0.8510	0.8571	0.7903	0.8285
CSQ-S	0.8661/**1.8%**	0.8732/**1.9%**	0.8034/**1.7%**	0.8318/**0.4%**
CSQ-C	0.8670/**1.9%**	0.8688/**1.4%**	0.8007/**1.3%**	0.8353/**0.8%**
DBDH	0.8440	0.8421	0.8122	0.8323
DBDH-S	0.8626/**2.2%**	0.8675/**3.0%**	0.8177/**0.7%**	0.8388/**0.8%**
DBDH-C	0.8658/**2.6%**	0.8731/**3.7%**	0.8135/**0.1%**	0.8380/**0.7%**

Single-Label Data **Multi-Label Data**

Performance Evaluation (MAP)

Retrieve k items MAP@k = Mean of Average Precisions from 1 to k (Area under PR Curve)

-S: Sliced Wasserstein Estimate | -C: Proposed Wasserstein Estimate

Method	CIFAR-10			NUS-WIDE			COCO		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
DSDH [40]	0.7909	0.8072	0.8278	0.8270	0.8455	0.8640	0.7331	0.7853	0.8074
DSDH-S	0.8187/**3.5%**	0.8439/**4.6%**	0.8517/**2.9%**	0.8282/**0.1%**	0.8461/**0.1%**	0.8712/**0.8%**	0.7330/*0.0%*	0.8030/**2.3%**	0.8404/**4.1%**
DSDH-C	0.8531/**7.9%**	0.8620/**6.8%**	0.8658/**4.6%**	0.8433/**2.0%**	0.8631/**2.1%**	0.8749/**1.3%**	0.7424/**1.3%**	0.8032/**2.3%**	0.8408/**4.1%**
HashNet [6]	0.6922	0.8311	0.8566	0.7728	0.8336	0.8654	0.6899	0.7666	0.8098
HashNet-S	0.8131/**17%**	0.8573/**3.2%**	0.8749/**2.1%**	0.8062/**4.3%**	0.8438/**1.2%**	0.8713/**0.7%**	0.7215/**4.6%**	0.7764/**1.3%**	0.8189/**1.1%**
HashNet-C	0.7939/**14%**	0.8467/**1.9%**	0.8691/**1.5%**	0.8002/**3.5%**	0.8437/**1.2%**	0.8791/**1.6%**	0.7202/**4.4%**	0.7789/**1.6%**	0.8202/**1.3%**
GreedyHash [50]	0.8223	0.8474	0.8646	0.7802	0.8081	0.8328	0.6533	0.7219	0.7561
GreedyHash-S	0.8280/**0.7%**	0.8497/**0.3%**	0.8653/**0.1%**	0.7815/**0.1%**	0.8083/*0.0%*	0.8390/**0.7%**	0.6668/**2.1%**	0.7291/**1.0%**	0.7618/**0.8%**
GreedyHash-C	0.8375/**1.9%**	0.8536/**0.7%**	0.8722/**0.9%**	0.7890/**1.1%**	0.8179/**1.2%**	0.8477/**1.8%**	0.6637/**1.6%**	0.7299/**1.1%**	0.7712/**2.0%**
DCH [5]	0.8302	0.8432	0.8558	0.8015	0.8061	0.8040	0.7578	0.7792	0.7723
DCH-S	0.8372/**0.8%**	0.8515/**1.0%**	0.8602/**0.5%**	0.8058/**0.5%**	0.8079/**0.2%**	0.8067/**0.3%**	0.7657/**1.1%**	0.7831/**0.5%**	0.7803/**1.0%**
DCH-C	0.8446/**1.7%**	0.8596/**1.9%**	0.8711/**1.8%**	0.8159/**1.8%**	0.8145/**1.0%**	0.8155/**1.4%**	0.7702/**1.6%**	0.7892/**1.3%**	0.7807/**1.1%**
CSQ [58]	0.8069	0.8291	0.8366	0.7992	0.8384	0.8596	0.6783	0.7550	0.8146
CSQ-S	0.8401/**4.1%**	0.8555/**3.2%**	0.8554/**2.3%**	0.8044/**0.7%**	0.8495/**1.3%**	0.8626/**0.4%**	0.7036/**3.7%**	0.7765/**2.8%**	0.8234/**1.0%**
CSQ-C	0.8457/**4.8%**	0.8558/**3.2%**	0.8652/**3.4%**	0.8054/**0.8%**	0.8511/**1.5%**	0.8701/**1.2%**	0.6989/**3.0%**	0.7752/**2.7%**	0.8255/**1.3%**
DBDH [60]	0.7660	0.8223	0.8492	0.8305	0.8552	0.8666	0.7202	0.7826	0.8042
DBDH-S	0.8458/**10%**	0.8587/**4.4%**	0.8603/**1.3%**	0.8387/**1.0%**	0.8577/**0.3%**	0.8680/**1.8%**	0.7461/**2.2%**	0.7996/**3.7%**	0.8336/**4.3%**
DBDH-C	0.8466/**10%**	0.8593/**4.5%**	0.8668/**2.1%**	0.8395/**1.1%**	0.8633/**0.9%**	0.8760/**1.1%**	0.7389/**2.6%**	0.7889/**0.8%**	0.8308/**3.9%**

Single-Label Data | **Multi-Label Data**

Summary

- Show that **better quantization** results in **better retrieval.**
- Learn better quantization with a **single loss.**
- Propose an **efficient divergence estimate** for single-loss.

Our approach can be used with any existing Deep Supervised Hashing techniques to learn better-quantized hash functions!

Thank You!

Contact: Khoa D. Doan
Website: khoadoan.me
Email: khoadoan106@gmail.com
Code: https://github.com/khoadoan106/single_loss_quantization