Unsupervised Quantization

Haowei Zhang

Outline

- Introduction
- Dataset and Evaluation Metrics
- PQ based Method
- RVQ based Method
- AQ based Method

Outline

- Introduction
- Dataset and Evaluation Metrics
- PQ based Method
- RVQ based Method
- AQ based Method

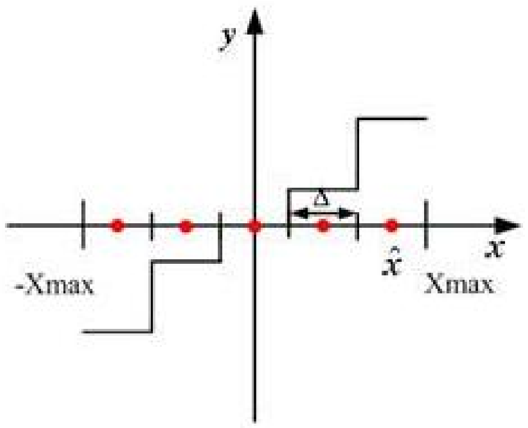
Introduction

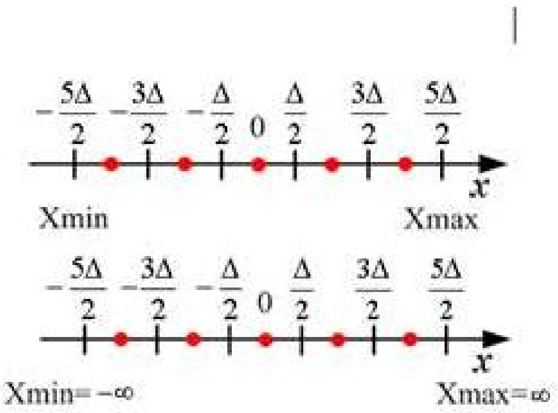
- What is quantization?

- Quantization is a process that maps all inputs within a specified range to a common value.

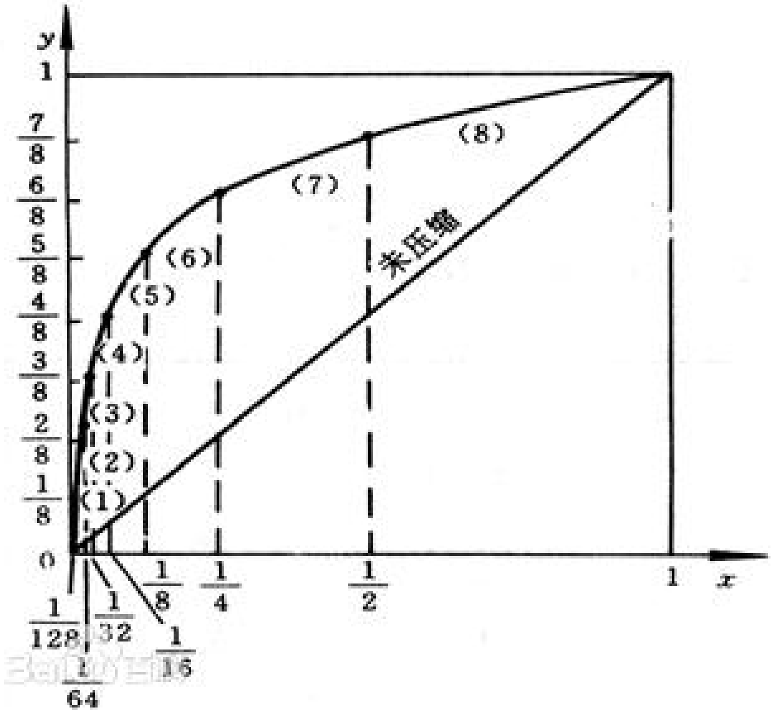
Introduction

- Scalar Quantization





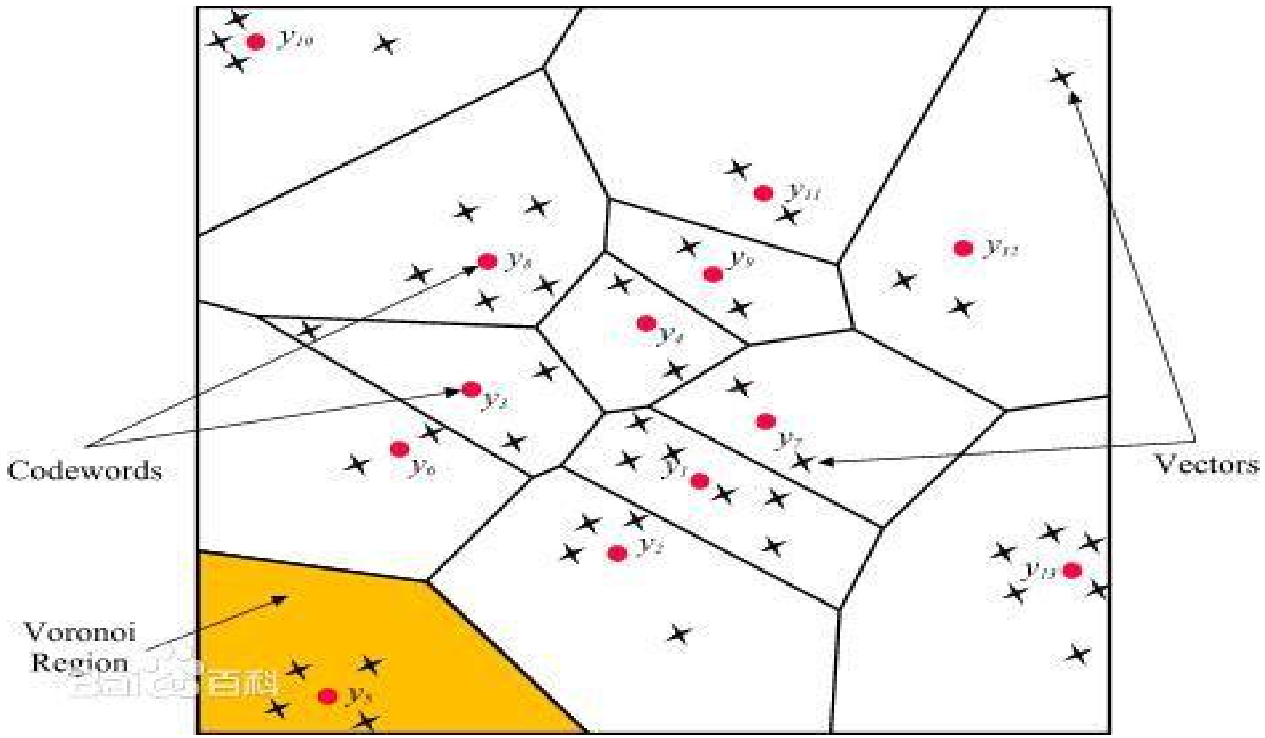
Uniform Quantization



Nonuniform Quantization
(A-law PCM)

Introduction

- Vector Quantization (VQ)
  - Learn from dataset
  - K-means clustering



Formulation

quantizer $q(\cdot): R^D \mapsto R^D$ $q(x) = \arg\min_{c_i \in C} d(x, c_i)$

encoder $i(\cdot): R^D \mapsto \{1,2,...,k\}$ $i(x) = \arg\min_{i \in \{1,...,k\}} d(x, c_i)$

Codebook $C = \{c_i\}_{i=1}^{k}$

Quantization Error $MSE(q) = \frac{1}{N}\sum_{i=1}^{N} \|x_i - q(x_i)\|_2^2$

Applications

- Lossy Compression
- Approximate Nearest Neighbor Search (ANN)
- K-Nearest Neighbor (KNN)

Quantization vs. Hashing

- Similarities
  - mapping high-dimensional vectors to short binary codes
- Differences
  - hashing use Hamming distance, faster
  - quantization use Euclidean distance, more accurate
  - quantization has extra memory consumption to store the codebook

Outline

- Introduction
- Dataset and Evaluation Metrics
- PQ based Method
- RVQ based Method
- AQ based Method

Datasets

Datasets	SIFT 1M	SIFT 1B	GIST 1M	Deep 1B
descriptor	SIFT	SIFT	GIST	Deep Feature(PCA)
dimensionality	128	128	960	96
learning set size	100,000	100,000,000	500,000	350,000,000
database set size	1,000,000	1,000,000,000	1,000,000	1,000,000,000
queries set size	10,000	10,000	1,000	10,000
groundtruth	k(e.g. k = 100) nearest neighbors in database for each query			

Evaluation Metrics

- recall@R : the proportion of query vectors for which the nearest neighbor is ranked in the first R positions
- MAP (Mean Average Precision)

Outline

- Introduction
- Dataset and Evaluation Metrics
- PQ based Method
- RVQ based Method
- AQ based Method

Limitations of VQ

- The quantization error is positively related to the total number of centroids k.

- Large k raises several issues:

1. large scale of training set (several times of k)

2. high training complexity O(TNkD)

3. huge memory consumption for storing the codebook O(kD)

Product Quantization (PQ)[1]

- key idea: split the original vector x into m distinct subvectors and quantize them separately

$$\underbrace{x_1, \ldots, x_{D^*}}_{u_1(x)}, \ldots, \underbrace{x_{D-D^*+1}, \ldots, x_D}_{u_m(x)} \rightarrow q_1\big(u_1(x)\big), \ldots, q_m\big(u_m(x)\big)$$

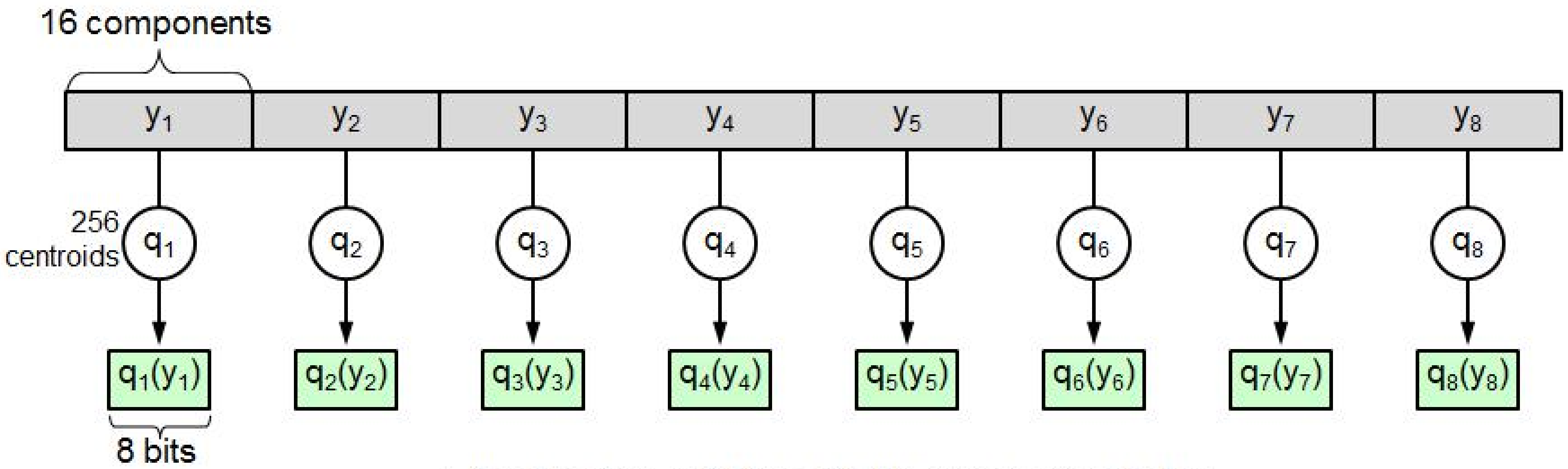
$$D^* = D/m$$

$q_j(\cdot)$: low-complexity quantizer that has k^* centroids

total number of centroids: $k = (k^*)^m$

Product Quantization (PQ)[1]

128D float vector => 64bit PQ code



⇒ 8 subvectors x 8 bits = 64-bit quantization index

Product Quantization (PQ)[1]

Method	codebook learning complexity	encoding complexity	codebook memory usage
VQ	TNkD	kD	kD
PQ	TNk^*D	k^*D	k^*D

typically $m = \{4, 8, 16\}$ $k^*=256$,
respectively $k = (k^*)^m = \{2^{32}, 2^{64}, 2^{128}\}$

Product Quantization (PQ)[1]

- How to search with PQ codes ?

key point : How to compute the distances between the query vector and the database vectors (stored as PQ codes) ?

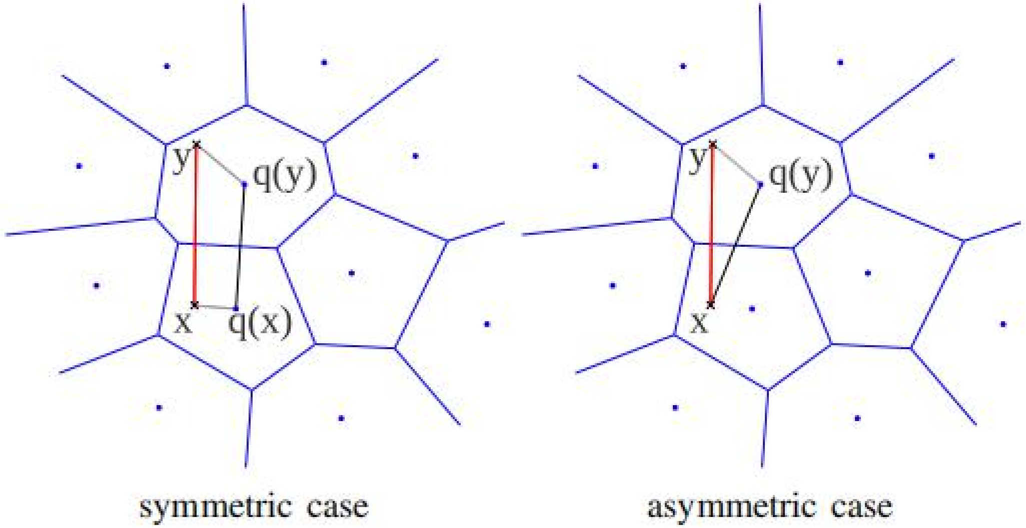
Product Quantization (PQ)[1]

Symmetric Distance Computation (SDC):

quantize both query x and database y

Asymmetric distance computation (ADC):

only quantize database y



ADC is more accurate than SDC

Product Quantization (PQ)[1]

- ADC formula

$$d(x,y)^2 \approx d(x,q(y))^2 = \sum_{j=1}^{m} d(u_j(x), q_j(u_j(y)))^2$$

for fixed query x, $d(u_j(x)), q_j(u_j(y)))^2$ has only k* possible values

pre-compute and store in lookup tables

time complexity: O(m)   momery usage: O(mk*)

Non Exhaustive Search

- a two step strategy

1. coarse quantization (rapid access to a small fraction of database)
    - train a quantizer (using k-means) with small k (e.g. k = 256)
    - compute residual $r(x) = x - q_c(x)$

2. product quantization
    - quantize the residual vector $q(x) = q_p(r(x))$

IVFADC[1]

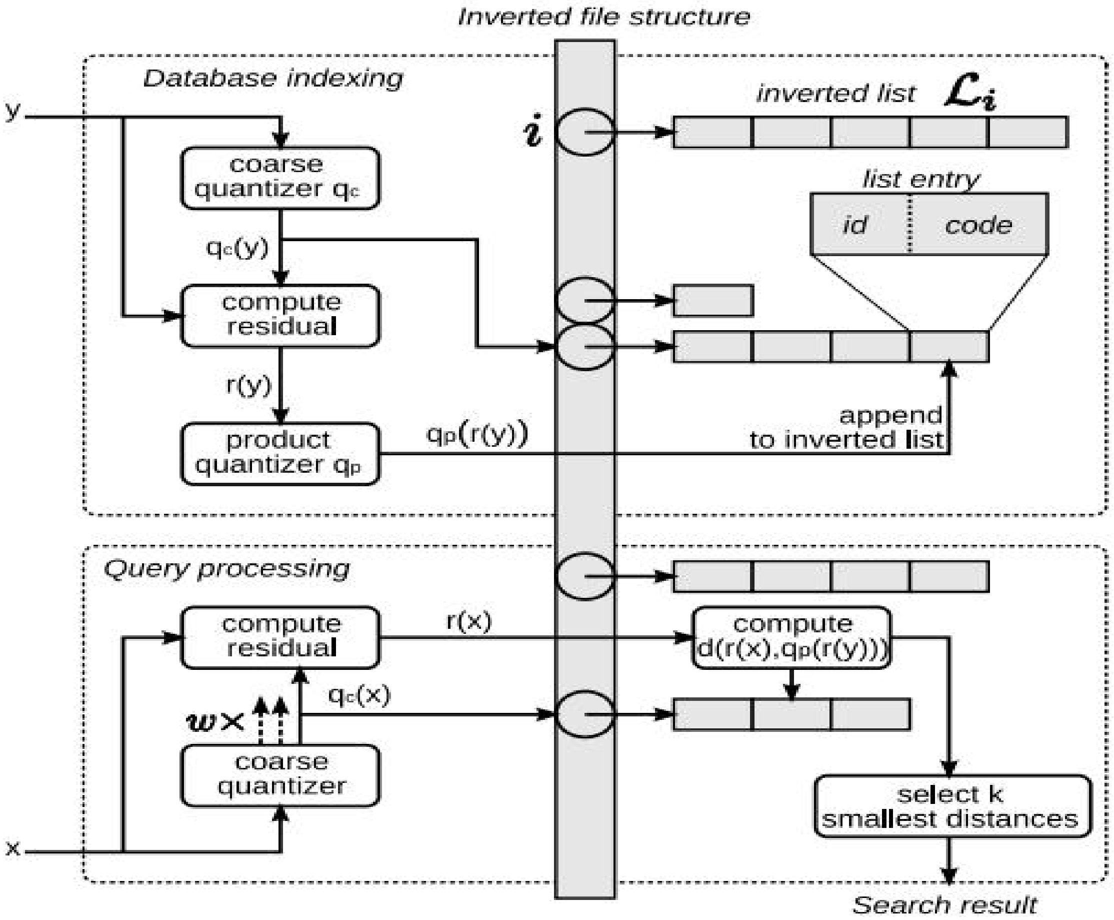


Fig. 5. Overview of the *inverted file with asymmetric distance computation* (IVFADC) indexing system. *Top*: insertion of a vector. *Bottom*: search.

Iterative Quantization (ITQ)[2]

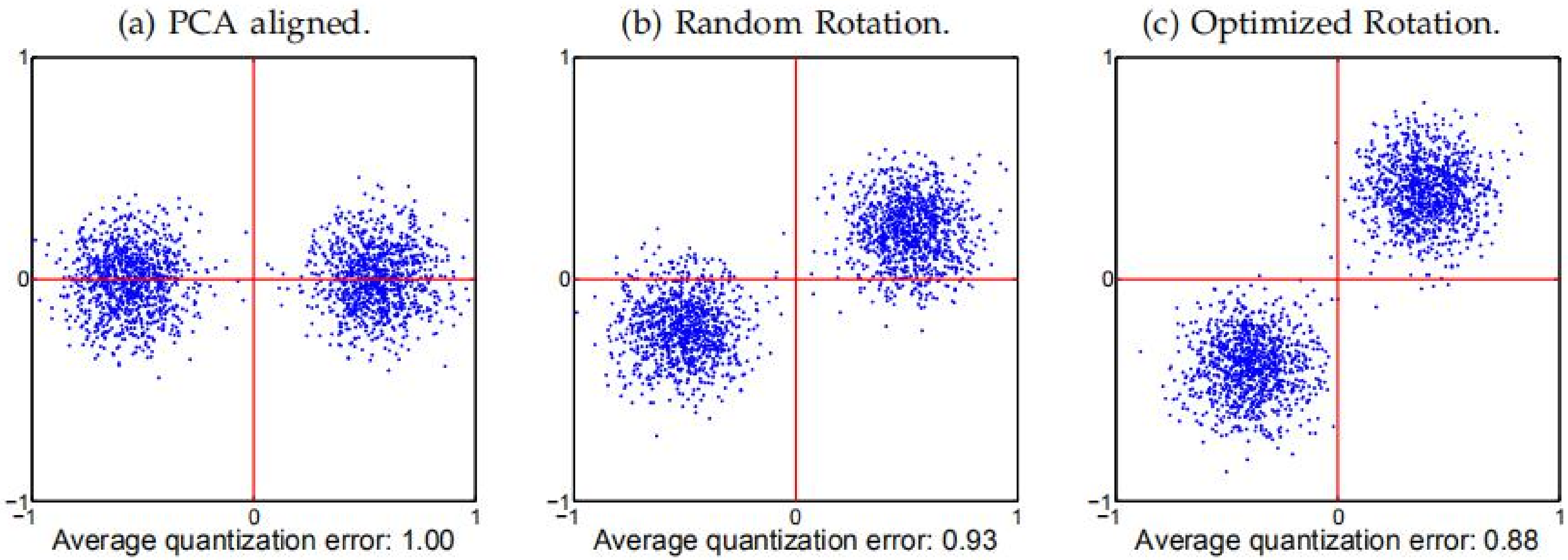
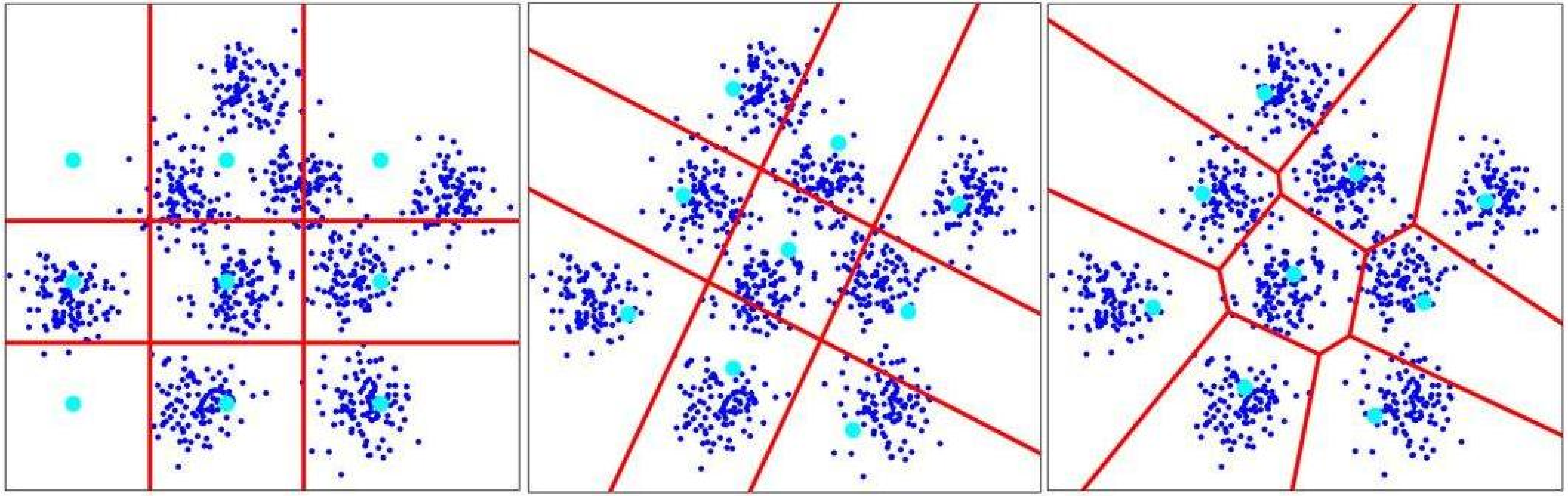


Figure 1. Toy illustration of our ITQ method (see Section 3 for details). The basic binary encoding scheme is to quantize each data point to the closest vertex of the binary cube, $(\pm1, \pm1)$ (this is equivalent to quantizing points according to their quadrant). (a) The x and y axes correspond to the PCA directions of the data. Note that quantization assigns points in the same cluster to different vertices. (b) Randomly rotated data – the variance is more balanced and the quantization error is lower. (c) Optimized rotation found by ITQ – quantization error is lowest, and the partitioning respects the cluster structure.

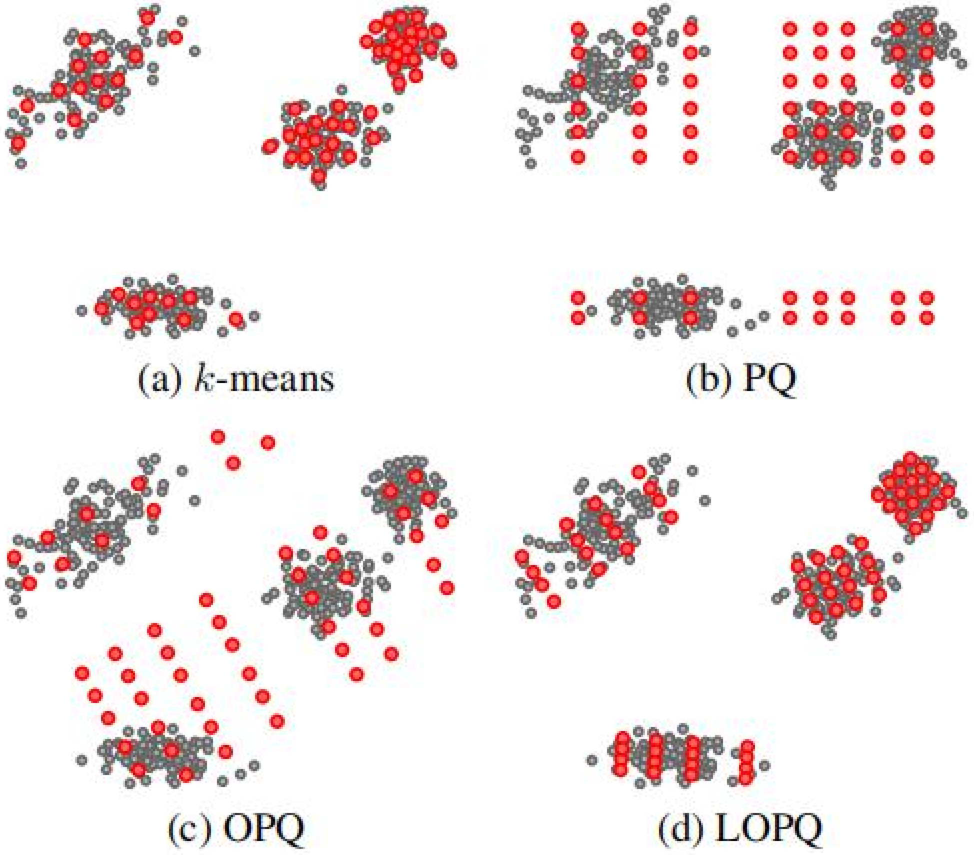
Optimized Product Quantization (OPQ)[3]

pre-rotation



PQ OPQ VQ

Locally Optimized Product Quantization (LOPQ)[4]



Experiments[4]

Method	$R = 1$	$R = 10$	$R = 100$
Ck-means [15]	–	–	0.649
IVFADC	0.106	0.379	0.748
IVFADC [13]	0.088	0.372	0.733
I-OPQ	0.114	0.399	0.777
Multi-D-ADC [3]	0.165	0.517	0.860
LOR+PQ	0.183	0.565	0.889
LOPQ	**0.199**	**0.586**	**0.909**

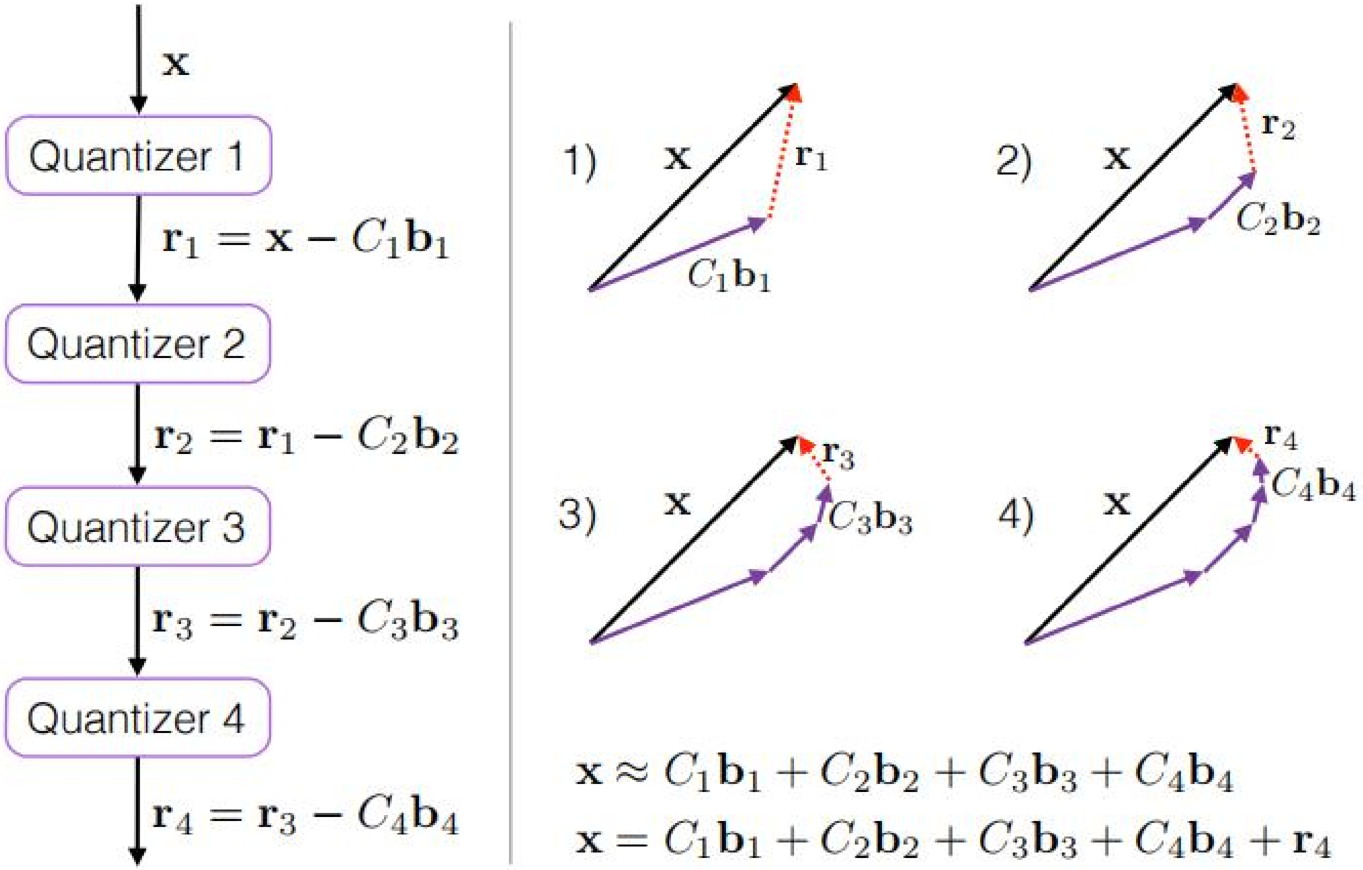
Table 1. Recall@$\{1, 10, 100\}$ on SIFT1B with 64-bit codes, $K = 2^{13} = 8192$ and $w = 64$. For Multi-D-ADC, $K = 2^{14}$ and $T = 100$K. Rows including citations reproduce authors' results.

Outline

- Introduction
- Dataset and Evaluation Metrics
- PQ based Method
- RVQ based Method
- AQ based Method

Residual Vector Quantization (RVQ)[5]

hierarchical structure



Transformed Residual Quantization (TRQ)[6]

rotation of the residual space

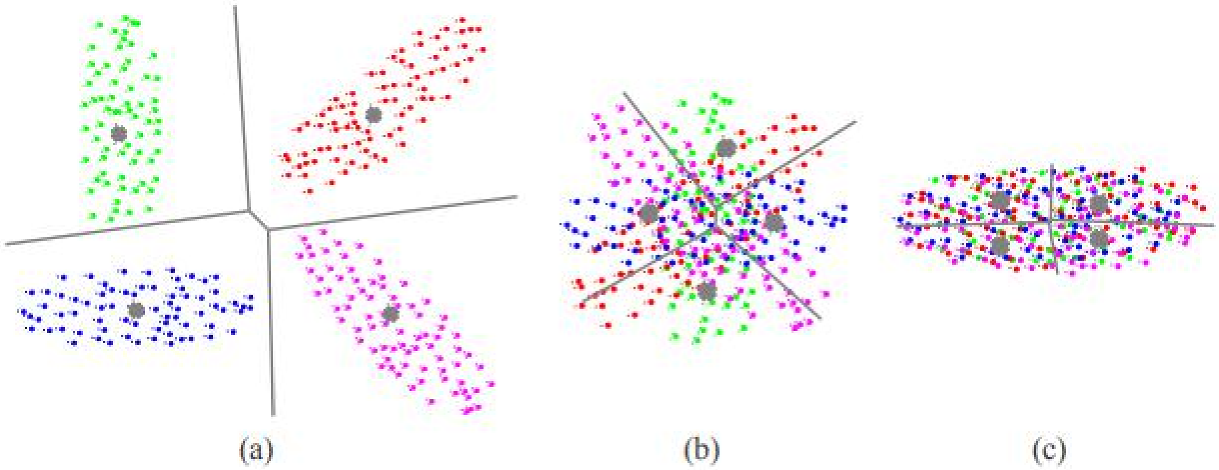


Fig. 1. A toy example of two-level RQ and TRQ. While the vector quantizer (a) is identical, the residual space in the RQ model (b) is much noisier than the TRQ one (c).

Competitive Quantization (CompQ)[7]

- the codebooks are jointly optimized using the Stochastic Gradient Decent (prevents overfitting in upper levels and increases the contribution of lower levels)

- apply Beam Search in encoding to avoid local optimum

Experiments[7]

TABLE 4
Test Results for SIFT1M, 64-bit Codes

	recall@1	recall@10	recall@100
RVQ	0.257	0.659	0.952
CKM/OPQ	0.243	0.638	0.940
APQ	0.298	0.741	0.972
CQ	0.288	0.716	0.967
OCK	0.274	0.680	0.945
ERVQ	0.276	0.694	0.962
OTQ	0.317	0.748	0.972
KSSQ	0.325	0.754	0.976
CompQ	**0.352**	**0.795**	**0.987**

Outline

- Introduction
- Dataset and Evaluation Metrics
- PQ based Method
- RVQ based Method
- AQ based Method

Additive Quantization (AQ)[8]

generalization of PQ

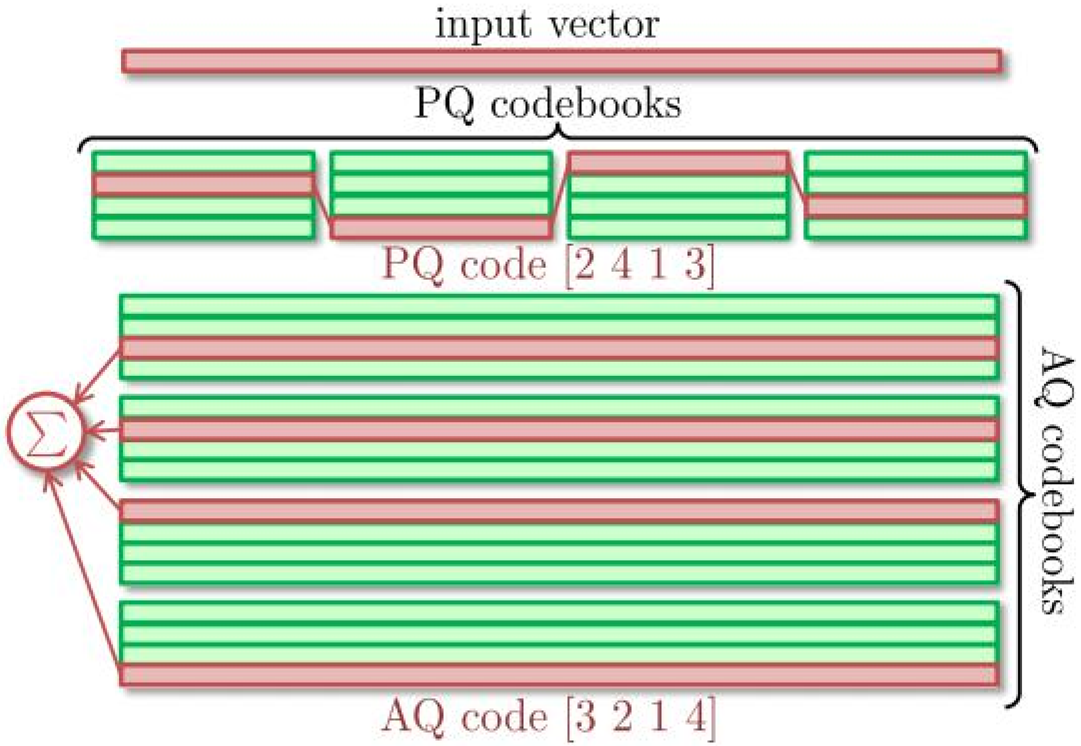


Figure 1. Product quantization (PQ) vs. Additive quantization

Additive Quantization (AQ)[8]

- encoding problem
  - find code b_i that minimizing $\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 = \|\mathbf{x} - \sum_{i}^{m} C_i \mathbf{b}_i\|_2^2$
  - k^m possible candidates (computational expensive for exhaustive search)
  - known as fully connected MRF(Markov Random Field) problem (is NP-hard)
  - approximate algorithm: Beam Search (slightly different from CompQ)

Additive Quantization (AQ)[8]

- search problem
  - ADC formula $\|\mathbf{x}-\hat{\mathbf{x}}\|_2^2 = \|\mathbf{x}-\sum_i^m C_i\mathbf{b}_i\|_2^2 = \|\mathbf{x}\|_2^2 - 2\cdot\sum_i^m\langle\mathbf{x}, C_i\mathbf{b}_i\rangle + \|\sum_i^m C_i\mathbf{b}_i\|_2^2$

    where $\|\sum_i^m C_i\mathbf{b}_i\|_2^2 = \sum_i^m\|C_i\mathbf{b}_i\|_2^2 + \sum_i^m\sum_{j\neq i}^m\langle C_i\mathbf{b}_i, C_j\mathbf{b}_j\rangle$
  - precompute $\langle\mathbf{x}, C_i\mathbf{b}_i\rangle$ $\langle C_i\mathbf{b}_i, C_j\mathbf{b}_j\rangle$
  - complexity $O(m^2)$
  - scalar quantize $\|\sum_i^m C_i\mathbf{b}_i\|_2^2$ to reduce search complexity to O(m)

Composite Quantization (CQ)[9]

- extra constraint : constant inter-dictionary-element-product

$$\min_{\{\mathbf{C}_m\},\{\mathbf{b}_n\},\epsilon} \sum_{n=1}^{N} \|\mathbf{x}_n - [\mathbf{C}_1\mathbf{C}_2\cdots\mathbf{C}_M]\mathbf{b}_n\|_2^2$$

$$\boxed{\text{s. t. } \sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M} \mathbf{b}_{ni}^T\mathbf{C}_i^T\mathbf{C}_j\mathbf{b}_{nj} = \epsilon}$$

$$\mathbf{b}_n = [\mathbf{b}_{n1}^T\mathbf{b}_{n2}^T\cdots\mathbf{b}_{nM}^T]^T$$

$$\mathbf{b}_{nm} \in \{0,1\}^K,\ \|\mathbf{b}_{nm}\|_1 = 1$$

$$n = 1,2,\cdots,N, m = 1,2,\cdots,M.$$

- also reduce search complexity from $O(m^2)$ to $O(m)$

Local Search Quantization (LSQ)[10]

- use Stochastic local search (SLS) for encoding

- alternatively do the following
  - perturbing the current solution s
  - performing local search starting from the perturbed solution, leading to a new candidate solution s'
  - acceptance criterion (whether to continue the search process from s or s')

Local Search Quantization (LSQ)[10]

- comparison with Beam Search

Beam Search $O(mh^2(m + \log mh))$

Local Search $O(m^2 h)$

typical values $m = \{4,8,16\}$ $h = 256$

in practice, Local Search is 30-50x faster than Beam Search

Experiments[10]

	SIFT1M – 64 bits		
	R@1	R@10	R@100
PQ	22.53 ± 0.31	60.14 ± 0.41	91.99 ± 0.17
OPQ	24.34 ± 0.52	63.89 ± 0.30	94.04 ± 0.08
AQ-7 [7]	26	70	95
CQ [11]	29	71	96
LSQ-16	$\underline{29.37} \pm 0.18$	$\underline{72.54} \pm 0.26$	$\underline{97.27} \pm 0.14$
LSQ-32	$\mathbf{29.79} \pm 0.26$	$\mathbf{73.12} \pm 0.20$	$\mathbf{97.49} \pm 0.09$

Experiments[11]

Table 4. Comparison between CompQ, LSQ and LSQ++ on SIFT1M using 64 bits.

	Iters	Init	Training	Base encoding	Total	R@1
CompQ [27] (C++)	250	–	–	–	38 h	0.352
LSQ [21] (Julia, CUDA)	25	1.1 m	2.8 m	29 s (32 iters)	4.4 m	0.340
LSQ++ (Julia, CUDA)	25	1.1 m	33 s	29 s (32 iters)	2.1 m	0.346
LSQ++ (Julia, CUDA)	50	2.2 m	1.1 m	58 s (64 iters)	4.3 m	0.348
LSQ++ (Julia, CUDA)	100	4.4 m	2.2 m	1.9 m (128 iters)	8.5 m	0.351
LSQ++ (Julia, CUDA)	100	4.4 m	2.2 m	3.9 m (256 iters)	10.5 m	<u>0.353</u>

Q&A

References

[1] H. J´egou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. IEEE TPAMI, 33(1):117–128, 2011.

[2] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. IEEE TPAMI, 35(12):2916–2929, 2013.

[3] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. IEEE TPAMI, 36(4):744–755, 2014.

[4] Y. Kalantidis and Y. Avrithis. Locally optimized product quan_x0002_tization for approximate nearest neighbor search. In Proc. IEEE CVPR, 2014.

[5] Y. Chen, T. Guan, and C. Wang. Approximate nearest neighbor search by residual vector quantization. Sensors, 10(12):11259– 11273, 2010.

[6] Jiangbo Yuan, and Xiuwen Liu. "Fast Nearest Neighbor Search with Transformed Residual Quantization." 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA) IEEE, 2016.

References

[7] E. C. Ozan, S. Kiranyaz, and M. Gabbouj. Competitive quan_x0002_tization for approximate nearest neighbor search. IEEE TKDE, 28(11):2884–2894, 2016.

[8] A. Babenko and V. Lempitsky. Additive quantization for extreme vector compression. In Proc. IEEE CVPR, 2014.

[9] T. Zhang, C. Du, and J. Wang. Composite quantization for ap_x0002_proximate nearest neighbor search. In Proc. ICML, 2014.

[10] J. Martinez, J. Clement, H. H. Hoos, and J. J. Little. Revisiting additive quantization. In Proc. ECCV, 2016.

[11] Julieta Martinez, Shobhit Zakhmi, Holger H. Hoos, James J. Little. "LSQ++: Lower running time and higher recall in multi-codebook quantization", ECCV 2018.