APPROXIMATE SEARCH WITH QUANTIZED SPARSE REPRESENTATIONS

Himalaya Jain[1,2], Patrick Pérez[1], Rémi Gribonval[2], Joaquin Zepeda[1] and Hervé Jégou[3]
[1]{firstname.lastname}@technicolor.com, [2]{firstname.lastname}@inria.fr, [3]rvj@fb.com

GOAL

Approximate nearest neighbor (ANN) search

I. CONTRIBUTIONS

- A novel quantization approach based on constrained sparse coding for ANN search.
- The proposed approach is demonstrated with an adaptation of two vector quantizers:
  - Product Quantizer (PQ) as "Qα-PQ"
  - Residual Vector Quantizer (RVQ) as "Qα-RVQ"
- Competitive results on four datasets including billion-sized BIGANN dataset.
  - Significant improvement over RVQ.
  - Faster encoding with similar or better accuracy.

II. BACKGROUND

- ***ANN search*** attempts to find a query's nearest neighbor efficiently by relying on compressed representations.
- ***A vector quantizer (VQ)*** maps a vector to a codeword,

$$\mathbf{x} \mapsto \mathbf{c}_{k(\mathbf{x})} \in \mathbf{C} \tag{1}$$

where $\mathbf{C}$ is a codebook containing K codewords. The codeword id of vector $\mathbf{x}$, $k(\mathbf{x}) \in [\![1, K]\!]$, is often given by:

$$k(\mathbf{x}) = \arg\min_{i \in [\![1,K]\!]} \|\mathbf{x} - \mathbf{c}_i\|^2 \tag{2}$$

for Euclidean based encoding, or for cosine similarity based encoding as:

$$k(\mathbf{x}) = \arg\max_{i \in [\![1,K]\!]} \mathbf{x}^\top \mathbf{c}_i \tag{3}$$

- ***Product quantizer (PQ)*** [2]

There are M sub-codebooks $\mathbf{C}^m$, each represents a different orthogonal sub-space.

$$Q(\mathbf{x}) = \sum_{m=1}^{M} \mathbf{c}^m_{k_m(\mathbf{x})} \tag{4}$$

where $\mathbf{c}^m_i \in \mathbf{C}^m$.
An encoded vector takes $M \times \log K$ bits.

- ***Residual vector quantizer (RVQ)*** [1]

The sub-codebooks form a hierarchy and each represents a different level of residuals.

$$Q(\mathbf{x}) = \sum_{m=1}^{M} \mathbf{c}^m_{k_m(\mathbf{r}_m)} \tag{5}$$

where $\mathbf{r}_m = \mathbf{x} - \sum_{j=1}^{m-1} \mathbf{c}^j_{k_j(\mathbf{r}_j)}$, with $\mathbf{r}_1 = \mathbf{x}$.
An encoded vector takes $M \times \log K$ bits.

- ***Sparse coding view of VQ***. In sparse coding a vector is represented as $\mathbf{C}\boldsymbol{\alpha}$ which is equivalent to

$$Q(\mathbf{x}) = \sum_{i=1}^{K} \alpha_i \mathbf{c}_i \tag{6}$$

where $\boldsymbol{\alpha}$ is a sparse vector.
PQ and RVQ can be seen as sparse encoding with binary weights and sparsity M (out of MK).

REFERENCES

[1] Approximate nearest neighbor search by residual vector quantization. Chen, Yongjian and Guan, Tao and Wang, Cheng. *Sensors, 2010.*
[2] Product quantization for nearest neighbor search. Jégou, Hervé and Douze, Matthijs and Schmid, Cordelia. *IEEE TPAMI, 2011.*
[3] The inverted multi-index. Babenko, Artem and Lempitsky, Victor. *CVPR, 2012.*
[4] All about VLAD. Arandjelovic, Relja and Zisserman, Andrew. *CVPR, 2013.*
[5] Cartesian k-means. Norouzi, Mohammad and Fleet, David J. *CVPR, 2013*
[6] Additive quantization for extreme vector compression. Babenko, Artem and Lempitsky, Victor. *CVPR, 2014.*
[7] Composite Quantization for Approximate Nearest Neighbor Search. Zhang, Ting. *ICML, 2014.*
[8] Optimized residual vector quantization for efficient approximate nearest neighbor search. Ai, Liefu and Yu, Junqing and Wu, Zebin and He, Yunfeng and Guan, Tao *Multimedia Systems, 2015.*

III. APPROACH

- ***Proposed representation***

$$Q(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m(\mathbf{x}) \mathbf{c}^m_{k_m(\mathbf{x})} \tag{7}$$

$\alpha_m(\mathbf{x}) \in \mathbb{R}$ and the codewords are unit norm vectors.

- Coefficient vectors formed as $\boldsymbol{\alpha} = [\alpha_1(\mathbf{x}), \ldots, \alpha_M(\mathbf{x})]$ are vector quantized.

$$\mathbf{x} \mapsto \sum_{m=1}^{M} \alpha_m(\mathbf{x}) \mathbf{c}^m_{k_m(\mathbf{x})}$$
$$\boldsymbol{\alpha} = [\alpha_1(\mathbf{x}), \ldots, \alpha_M(\mathbf{x})]$$
Vector Quantize
$$\mathbf{a}_{p(\boldsymbol{\alpha})}$$
$$Q(\mathbf{x}) = [\mathbf{c}^1_{k_1(\mathbf{x})}, \ldots, \mathbf{c}^M_{k_M(\mathbf{x})}]\mathbf{a}_{p(\boldsymbol{\alpha})}$$

where $\mathbf{a}_{p(\boldsymbol{\alpha})} \in \mathbf{A}$, the coefficient codebook, with P codewords and $P(\boldsymbol{\alpha}) \in [\![1, P]\!]$

- ***Qα-PQ***

$$k_m(\mathbf{x}) = \arg\max_{i \in [\![1,K]\!]} \mathbf{x}^\top \mathbf{c}^m_i$$
$$\alpha_m(\mathbf{x}) = \mathbf{x}^\top \mathbf{c}^m_{k_m(\mathbf{x})}$$
$$\boldsymbol{\alpha} \mapsto \mathbf{a}_{p(\boldsymbol{\alpha})}$$

$\mathbf{x}$ is encoded as $\left([k_1(\mathbf{x}), \ldots, k_M(\mathbf{x})], \mathbf{a}_{p(\boldsymbol{\alpha})}\right)$

- ***Qα-RVQ***

$$k_m(\mathbf{r}_m) = \arg\max_{i \in [\![1,K]\!]} \mathbf{r}_m^\top \mathbf{c}^m_i$$
$$\alpha_m(\mathbf{r}_m) = \mathbf{r}_m^\top \mathbf{c}^m_{k_m(\mathbf{x})}$$
$$\boldsymbol{\alpha} = [\mathbf{c}^1_{k_1(\mathbf{x})}, \ldots, \mathbf{c}^M_{k_M(\mathbf{x})}]^+ \mathbf{x}$$
$$\boldsymbol{\alpha} \mapsto \mathbf{a}_{p(\boldsymbol{\alpha})}$$

$\mathbf{x}$ is encoded as $\left([k_1(\mathbf{r}_1), \ldots, k_M(\mathbf{r}_M)], \mathbf{a}_{p(\boldsymbol{\alpha})}\right)$
where, $\mathbf{r}_m = \mathbf{x} - \sum_{j=1}^{m-1} \alpha_j(\mathbf{r}_j) \mathbf{c}^j_{k_j(\mathbf{r}_j)}$ and $\mathbf{r}_1 = \mathbf{x}$

- Learning $\mathbf{C}^m$: Spherical K-means.
- Learning $\mathbf{A}$: K-means.
- An encoded vector takes $M \times \log K + \log P$ bits.

IV. RESULTS: ANN SEARCH

- ***Impact of 1-byte quantization of $\boldsymbol{\alpha}$.***
  - Accuracy with quantized $\boldsymbol{\alpha}$ compared to unquantized $\boldsymbol{\alpha}$ and the associated VQ.
  - Qα-RVQ retains the accuracy of α-RVQ.
  - Qα-PQ performs well for lower M.

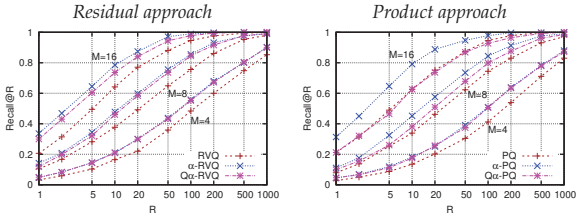

Figure 1: Recall rates on VLAD500K [4] (128 dimensional).

- ***Comparison with state-of-the-art.***
  - Accuracy of various quantization methods using the same compression rate.
  - Qα-RVQ performs the best after additive quantization(AQ) [6], a highly accurate but not scalable approach.

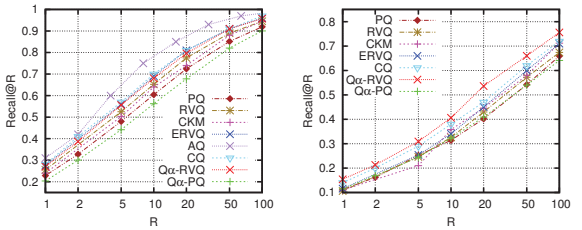

Figure 2: Recall rates on SIFT1M [2] & GIST1M [2] (64-bit encoding) compared with CKM [5], ERVQ [8], AQ [6] and CQ [7].

- ***Efficiency***
  - Comparing complexity of codebook learning, database encoding and ANN search.

In the above experiments, we set $P = 256$ and $K = 256$, except for Qα-PQ in Fig. 2 and Tab. 1 where K=128.

	PQ	Qα-PQ	RVQ	Qα-RVQ
learn	1	**0.212**	1.250	**0.719**
encode	1	**0.206**	1.347	**0.613**
search	1	1.867	1.220	1.909

Table 1: Complexity w.r.t. PQ on SIFT1M (64-bit encoding).

V. RESULTS: NON-EXHAUSTIVE ANN SEARCH

- ***Results on Billion-sized BIGANN [Jégou et al., ICASSP 2011] dataset using two inverted indexing methods.***

- ***Inverted index*** [2] uses a coarse quantizer and then the residual is quantized using the chosen encoder.
  - Qα-RVQ is the most accurate method.
  - The hierarchical structure of Qα-RVQ permits to further narrow the exhaustive search leading to much faster search.

Method (*bytes*)	R@1	R@10	R@100	time
PQ-64 (8)	0.111	0.388	0.756	1.00
PQ-72 (9)	0.144	0.471	0.825	1.03
RVQ (9)	0.124	0.421	0.803	1.02
Qα-PQ (9)	**0.139**	**0.450**	**0.811**	1.69
Qα-RVQ (10)	**0.160**	**0.514**	**0.868**	1.72
Qα-RVQ$_{128}$	0.160	0.514	0.868	**0.89**
Qα-RVQ$_8$	0.151	0.467	0.730	0.17

Table 2: Performance on BIGANN with relative timings.

- ***Inverted multi-index*** [3] uses two-fold product quantizers with a large number of inverted lists.
  - Qα-RVQ and Qα-PQ outperform others in accuracy.
  - Interestingly Qα-RVQ can be more accurate and faster than PQ by scanning fewer candidates.

	T =100K				T =30K			
Method (*bytes*)	R@1	R@10	R@100	time	R@1	R@10	R@100	time
PQ-64 (8)	0.170	0.535	0.869	29	0.170	0.526	0.823	11
Qα-PQ (9)	0.200	0.587	0.898	30	0.198	0.572	0.848	11
RVQ (9)	0.181	0.553	0.877	37	0.180	0.542	0.831	14
Qα-RVQ (10)	0.227	0.630	0.920	37	0.225	0.613	0.862	14

Table 3: Performace with T candidates exhaustively scanned. Time is in milliseconds.