Unsupervised Neural Quantization for Compressed-Domain Similarity Search

Stanislav Morozov
Yandex,
Lomonosov Moscow State University
stanis-morozov@yandex.ru

Artem Babenko
Yandex,
National Research University
Higher School of Economics
artem.babenko@phystech.edu

Abstract

We tackle the problem of unsupervised visual descriptors compression, which is a key ingredient of large-scale image retrieval systems. While the deep learning machinery has benefited literally all computer vision pipelines, the existing state-of-the-art compression methods employ shallow architectures, and we aim to close this gap by our paper. In more detail, we introduce a DNN architecture for the unsupervised compressed-domain retrieval, based on multi-codebook quantization. The proposed architecture is designed to incorporate both fast data encoding and efficient distances computation via lookup tables. We demonstrate the exceptional advantage of our scheme over existing quantization approaches on several datasets of visual descriptors via outperforming the previous state-of-the-art by a large margin.

1. Introduction

Unsupervised compression of high-dimensional visual descriptors has a long history in the computer vision community[33, 14]. Nowadays, the development of effective compact representations becomes even more crucial for the scalability of modern search engines, given the enormous amount of visual data in the Web.

Currently the dominant unsupervised compression methods[14, 6, 24, 2, 37, 22, 23] belong to the multi-codebook quantization (MCQ) paradigm. In this paradigm, descriptors are effectively approximated by a sum or a concatenation of a few *codeword* vectors, coming from several disjoint codebooks. Such a simple form of approximation enables the efficient computation of distances from uncompressed queries to compressed database vectors via the usage of lookup tables. While originally appeared for the image retrieval problem, the quantization methods are extensively used to increase the efficiency in a wide range of tasks, e.g. CNN compression[8, 36] or localization[19]. Indeed, the development of more advanced quantization methods remains an important research direction as they would benefit a whole range of large-scale computer vision applications.

Despite the ubiquitous use of deep architectures in different areas of computer vision, the unsupervised quantization for the compressed-domain retrieval, which we tackle in this paper, did not benefit from their power yet. While several recent works investigate the usage of deep architectures for the supervised MCQ scenario[35, 11, 18], the state-of-the-art unsupervised methods[22, 25, 23] remain shallow. Moreover, the recent work[27] has shown that even for the supervised compression problem, the usage of unsupervised MCQ outperforms several strong supervised baselines. To the best of our knowledge, at the moment it is not clear if the deep learning machinery can benefit the unsupervised quantization approaches. This is the central question we aim to answer in our paper.

As the main novelty, we introduce a new **Unsupervised Neural Quantization (UNQ)** method, which learns a nonlinear multi-codebook quantization model, trainable via SGD. Our model is partially inspired by the ideas from the recent works on generative modeling with discrete hidden variables[12, 21, 32], which, as we show, appear to be a natural fit for the compressed-domain retrieval problem. In its essence, UNQ works via embedding both codewords and data vectors into a common learnable vector space, where efficient nearest neighbor retrieval is possible. As we show in the experimental section, the non-linear nature of our model allows increasing the retrieval accuracy, compared to the existing shallow competitors.

Overall, the main contributions of our paper can be summarized as follows:

1. We propose a new method for the unsupervised quantization of visual descriptors. To the best of our knowledge, our method is the first successful case of the usage of deep architectures for the unsupervised MCQ for compressed-domain retrieval.

2. With the extensive experimental evaluation, we show

that the proposed method outperforms the existing techniques in terms of retrieval performance. For most operating points our method provides a new state-of-the-art on two common benchmarks.

3. The Pytorch implementation of the proposed method is available online[1].

The rest of the paper is organized as follows. In Section 2 we review the existing unsupervised quantization approaches. The proposed Unsupervised Neural Quantization model is described in Section 3 and experimentally evaluated in Section 4. Section 5 concludes the paper.

2. Related work

In this section we briefly review the main ideas from the previous works that are relevant for our method.

High-dimensional data compression. The existing methods for the high-dimensional data compression mostly fall into two separate lines of research. The first family[33, 7, 9] includes binary hashing methods, which map the original vectors into the Hamming space such that nearby vectors are mapped into hashes with small Hamming distances. The practical advantage of binary hashing is that it heavily benefits from the efficient binary computations in modern CPU architectures. The second family of methods generalizes the idea of vector quantization, and we refer to these methods as multi-codebook quantization (MCQ). MCQ methods typically do not involve the information loss on the query side, hence they typically outperform the binary hashing methods by a large margin. The state-of-the-art compression accuracy is currently achieved by the recent MCQ methods[25, 23] and in this paper we aim to improve their quality even further via the power of deep architectures.

Product quantization (PQ)[14] is a pioneering method from the MCQ family, which inspired further research on this subject. PQ encodes each vector $x \in \mathbf{R}^D$ as a concatenation of M codewords from M $\frac{D}{M}$-dimensional codebooks $C_1, \dots, C_M$, each containing K codewords. In other words, PQ decomposes a vector into M separate subvectors and applies vector quantization (VQ) to each subvector, while using a separate codebook. As a result each vector x is encoded by a tuple of codewords indices $[i_1, \dots, i_M]$ and approximated by $x \approx [c_{1i_1}, \dots, c_{Mi_M}]$. Fast Euclidean distance computation becomes possible via efficient ADC procedure[14] using lookup tables:

$$\|q - x\|^2 \approx \|q - [c_{1i_1}, \dots, c_{Mi_M}]\|^2 = \sum_{m=1}^{M} \|q_m - c_{mi_m}\|^2 \tag{1}$$

Method	(O)PQ	AQ/LSQ	UNQ
Compression Quality	Medium	High	High
Encoding complexity	Low	High	Low
Learning complexity	Low	High	High

Table 1. The qualitative comparison of Unsupervised Neural Quantization (UNQ) with the existing quantization methods.

where q_m — mth subvector of a query q. This sum can be calculated in M additions and lookups given that distances from query subvectors to codewords are precomputed.

From the geometry viewpoint, PQ effectively partitions the original vector space into K^M cells, each being a Cartesian product of M lower-dimensional cells. Such product-based approximation works better if the $\frac{D}{M}$-dimensional components of vectors have independent distributions. The degree of dependence is affected by the choice of the splitting, and can be further improved by orthogonal transformation applied to vectors as preprocessing. Two subsequent works have therefore looked into finding an optimal transformation [6, 24]. The modification of PQ corresponding to such pre-processing transformation is referred below as Optimized Product Quantization (OPQ).

Non-orthogonal quantizations. Several works [5, 2, 37, 22, 25, 23] generalize the idea of Product Quantization by approximating each vector by a sum of M codewords instead of concatenation. In this case, the ADC procedure is still efficient while the approximation accuracy is increased.

The first approach, Residual Vector Quantization [5], quantizes original vectors, and then iteratively quantizes the approximation residuals from the previous iteration. Another approach, Additive Quantization (AQ) [2], is the most general as it does not impose any constraints on the codewords from the different codebooks. Usually, AQ provides the smallest compression errors, however, it is much slower than other methods, especially for large M. Composite Quantization (CQ) [37] learns codebooks with a fixed value of scalar product between the codewords from different codebooks. Several recent works[22, 23, 25] elaborate the idea of Additive Quantization, proposing the more effective procedure for codebooks learning. Currently, state-of-the-art compression accuracy is achieved by the LSQ method[23]. We present the qualitative comparison of the existing MCQ methods with the open-source implementations as well as the proposed UNQ method in Table 1.

Compression with DNN. Several recent works[35, 11, 18] investigate the usage of deep architectures for multi-codebook quantization in the supervised compression scenario. In contrast, we tackle the more challenging unsupervised setup, where only shallow quantization methods are currently in use. To the best of our knowledge, there is only one recent paper[26] that employs a deep architec-

[1] https://github.com/stanis-morozov/unq

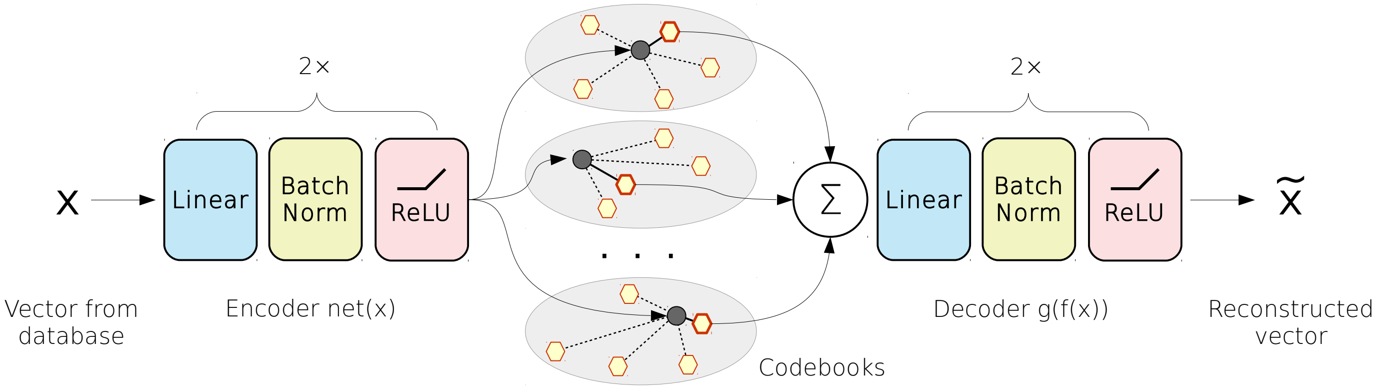

Figure 1. The proposed Unsupervised Neural Quantization model architecture. The encoder(left) maps data vector into a product of learned spaces (middle), selects codewords and decodes them back into the original vector space (right). The grey ellipses represent the codebook spaces and the orange hexagons denote the codewords in those codebooks.

ture for the unsupervised compression problem, but it does not work in the MCQ paradigm. Instead, [26] performs neighborhood-preserving mapping to a sphere with an additional ”spreading” regularizer that enforces the uniform distribution of mapped data points. Then [26] uses the fixed predefined lattices to quantize the data vectors. In our experiments, we demonstrate the advantage of UNQ over [26] in most operating points.

At the same time, several recent studies on generative modeling developed efficient ways to learn discrete representations with deep neural networks. One branch of such methods relies on continuous noisy relaxations of the discrete variables that can be trained by backpropagation [12, 21, 31]. Another popular approach to learning discrete variables is featured in Vector-Quantized Variational Autoencoder model[32, 17]. Instead of continuous relaxation, this approach uses straight-through gradient estimation to propagate gradient through discrete variables. To the best of our knowledge, our work is the first that experimentally demonstrates that with the appropriate training protocol, the discrete hidden variables can be successfully used for compressed-domain retrieval in the unsupervised scenario. While we acknowledge the existence of the recent concurrent preprint[34], which studies the close ideas, its experimental evaluation shows the performance of their approach to be on par with binary hashing methods, which are improper baselines. In contrast, our UNQ approach substantially outperforms the current state-of-the-art, as will be shown in experiments. While in this paper our approach is used only for compression of precomputed descriptors, the proposed architecture can be combined with existing self-supervised methods[4] for end-to-end unsupervised image compression.

3. Unsupervised Neural Quantization

We now introduce notation and discuss the proposed UNQ method in detail. Below, we always assume that image descriptors are vectors from the Euclidean space $\mathbf{R}^D$.

3.1. Motivation

All the existing quantization methods contain two essential modules: the *encoder* and the *distance function*.

The encoder $f(x) : \mathbf{R}^D \to \{1, \dots, K\}^M$ maps a data vector x into a tuple of M indices $\boldsymbol{i} = [i_1, \dots, i_M]$. In turn, the distance function $d(q, \boldsymbol{i}) : \mathbf{R}^D \times \{1, \dots, K\}^M \to \mathbf{R}$ estimates how far a query q is from an encoded database vector $f(x)$. Both $f(\cdot)$ and $d(\cdot, \cdot)$ typically depend on learnable parameters (e.g. the quantization codebooks in PQ or the rotation matrix in OPQ).

Currently, the state-of-the-art unsupervised quantization methods use shallow encoders and distance functions. In this study, we instead propose to use deep parametric models for both $f(\cdot)$ and $d(\cdot, \cdot)$ that are jointly trained to perform nearest neighbor retrieval.

3.2. Model

The architecture of our model is schematically presented on Figure 1. There are two main parts: the encoder maps the data vector x into a tuple of discrete codes, and the decoder reconstructs the original vector from its compressed representation.

In the encoder part, we use a simple feedforward neural network $net(x)$ with M output ”heads”, which are trained jointly. As will be shown below, one can think of $net(x) = [net(x)_1, \dots, net(x)_M]$ as a mapping of data vectors into a product of M learned spaces. Each of the spaces in this product posesses a codebook of K codewords and we denote by c_{mk} the codeword k from the codebook $m, m \in \{1, \dots, M\}, k \in \{1, \dots, K\}$.

We use this model to define the stochastic encoding function that assigns data vectors to discrete codes based on the dot product in the learned space. In particular, the probability of being encoded by the k-th code from m-th codebook is defined as:

$$p(c_{mk}|x) = \frac{\exp\left(\langle net(x)_m, c_{mk}\rangle/\tau_m\right)}{\sum\limits_j \exp\left(\langle net(x)_m, c_{mj}\rangle/\tau_m\right)} \tag{2}$$

Here $\tau_m \in (0, \infty)$ defines the temperature (inverse ”peakyness”) of the probability distribution. We treat both c_{mk} and τ_m as regular model parameters and optimize them with backpropagation.

Assuming conditional independence between the probabilities of codewords in different codebooks for a given x, we get:

$$p(c_1, ..., c_M|x) = \prod_{i=m}^{M} p(c_m|x) \tag{3}$$

We can now define our encoding function by maximizing over those probabilities:

$$\begin{aligned} f(x) &= \operatorname*{argmax}_{c_1,\ldots,c_M} p(c_1, ..., c_M|x) = \\ &= [\operatorname*{argmax}_{c_{1k}} p(c_{0k}|x), ..., \operatorname*{argmax}_{c_{Mk}} p(c_{Mk}|x)] = \\ &= [\operatorname*{argmax}_{c_{1k}} \langle net(x)_1, c_{1k}\rangle, ..., \operatorname*{argmax}_{c_{Mk}} \langle net(x)_M, c_{Mk}\rangle] \end{aligned} \tag{4}$$

However, in order to train our model we require encoder to be differentiable. Inspired by[29], we use a differentiable approximation of $f(x)$ using Gumbel-Softmax[12, 21] trick. The differentiable approximation $\tilde{f}_m(x)$ for m-th codebook is a stochastic function that maps x to a vector:

$$\tilde{f}(x)_m = softmax\{\log p(c_{mj}|x) + z_j, j{=}1..K\} \tag{5}$$

In the formula above, z_j is a sample from the standard Gumbel distribution that can be obtained as $z_j = -\log(-\log Uniform(0, 1))$. Note that the original Gumbel-Softmax distribution[12] divides all exponent rates by a small temperature factor, which we set to 1 in all our experiments.

While the relaxation $\tilde{f}(x)_m$ is differentiable, it does not produce one-hot vectors, which are needed for quantization methods. Hence during training we discretize $\tilde{f}(x)_m$ via using $argmax$ instead of $softmax$ in (5), which results in one-hot vector, corresponding to the index of the chosen codeword. While this discretization is not differentiable, we backpropagate through it using straight-through gradient estimation: the gradients w.r.t. function outputs are passed to its inputs with no transformation applied.

Then we feed M one-hot vectors, produced by the encoder, into decoder $g(\cdot)$: another feedforward network that adds the corresponding codewords and reconstructs the vector $\tilde{x}$ in the original data space.

In all our experiments, both $net(\cdot)$ and $g(\cdot)$ are simple fully-connected neural networks with ReLU activation functions and Batch Normalization[10] layers before each activation (see Figure 1). We describe the particular choice of $net(\cdot)$ and $g(\cdot)$ in the experimental section.

3.3. Nearest Neighbor Search

The retrieval of nearest neighbors in a quantized database is performed via the exhaustive search with distance function $d(q, \boldsymbol{i})$:

$$\operatorname*{argmin}_{\boldsymbol{i}} d(q, \boldsymbol{i}) \tag{6}$$

In our model, we use two different definitions for $d(q, \boldsymbol{i})$ to provide both high compression accuracy and efficient retrieval.

The first ”naive” distance function reconstructs the database vector with the decoder g and measures distance in the original data space:

$$d_1(q, \boldsymbol{i}) = \|q - g(\boldsymbol{i})\|_2^2 \tag{7}$$

However, the usage of d_1 for exhaustive search would require applying the decoder network to the whole database, inducing a prohibitively large computational cost for large databases.

Alternatively, one can define the distance function in the learned space of codebooks. This definition relies on the fact that both query mapping $net(q)$ and codewords belong to a shared space. This space is conveniently equipped with a dot-product-based probability (2) of picking a particular codeword.

Our intuition is that the nearest neighbor of data point q should have codewords that are likely to be assigned to q itself. In other words, in order to search for the nearest neighbors of q we want to consider candidates with the highest $p(c_1, ..., c_M|q)$. This naturally leads us to the following distance function:

$$\begin{aligned} d_2(q, \boldsymbol{i}) &= d_2(q, \{i_1, \ldots, i_M\}) = -logp(c_{1i_1}, ..., c_{Mi_M}|q) = \\ &= -\sum_{m=1}^{M}\left(\langle net(q)_m, c_{mi_m}\rangle - \log\sum_{k=1}^{K}\exp\langle net(q)_m, c_{mk}\rangle\right) = \\ &= -\sum_{m=1}^{M}\langle net(q)_m, c_{mi_m}\rangle + const(q) \end{aligned} \tag{8}$$

Compared to (7), the second formulation allows for an efficient search algorithm via lookup tables. First, the algorithm computes the dot products of $net(q)$ with the codewords in all codebooks using one pass of the encoder network and $O(M \cdot K)$ dot product computations. The algorithm can then find the nearest neighbor by iterating over the encoded data points and summing the cached distances, doing only M additions per database vector.

The search based on distance function (8) can be seen as a generalization of existing quantization methods. However, unlike [6, 22], the distance is computed not in the original data space but in a new learned space that is obtained via SGD training.

In practice, we combine both distance functions in a two-stage search: at first, we efficiently select L nearest candidates based on d_2, and then re-rank those candidates using the more expensive d_1. The additional reranking stage does not influence the total scheme efficiency by much, as only a small fraction of the database is reranked.

Of course, the existing shallow methods can also benefit from the additional reranking with DNN and we compare our scheme with this baseline in the experiments. Our experiments below demonstrate that the post-search reranking slightly increases the accuracy of shallow MCQ methods, but the overall performance of UNQ is substantially higher.

3.4. Training

We train our model by explicitly fitting the two distance functions to maximize recall. The first distance function is defined in the original data space and can be trained with autoencoder-like objective:

$$L_1 = \frac{1}{n} \sum_{x_i} d_1(x_i, \tilde{x_i}) = \frac{1}{n} \sum_{x_i} \left\| x_i - g(\tilde{f}(x_i)) \right\|_2^2 \quad (9)$$

However, there is no guarantee that minimizing this objective would result in good candidates being selected for the reranking stage. Therefore, we also need to train $d_2(\cdot, \cdot)$ with another objective term.

We employ a metric learning approach by minimizing the triplet loss in the learned space. Intuitively, we want x to be closer to it's true nearest neighbor x_+ than to the negative example x_-.

$$L_2 = \frac{1}{n} \sum_{x} max(0, \delta + d_2(x, f(x_+)) - d_2(x, f(x_-))) \quad (10)$$

Similarly to [26], we sample x_+ from top-3 true nearest neighbors of data point x. In turn, x_- is sampled uniformly from between 100-th to 200-th nearest neighbors, excluding x itself and three candidates for x_+. Following a popular practice from the metric learning field, we sample those vectors at the offset of each training epoch.

The final term for our objective is a regularizer for Gumbel-Softmax that encourages equal frequency of codewords. A common problem of nearly all methods for learning discrete variables is that they converge to poor local optima where some codes are (almost) never used. In order to alleviate this issue, we repurpose the squared Coefficient of Variation regularizer that was originally proposed in [28] to combat a similar imbalance in the Mixture of Experts layers.

The coefficient of variation is computed from codeword probabilities averaged over the training batch X:

$$p_{avg}(i_m|X) = \frac{1}{n} \sum_{x_i \in X} p(i_m|x_i)$$

$$CV^2(i_m) = \frac{Var[p_{avg}(i_m|X)]}{[E[p_{avg}(i_m|X)]]^2} \quad (11)$$

Our final objective is just a sum of those three terms with coefficients. In our experiments we pick α from $\{0.1, 0.01, 0.001\}$ via grid search. As for β, we decrease it linearly from 1.0 to 0.05 during training.

$$L = L_1 + \alpha \cdot L_2 + \beta \cdot \frac{1}{M} \sum_{m=1}^{M} CV^2(i_m) \quad (12)$$

The model is trained to minimize the training objective L using minibatch gradient descent with the recent Quasi-Hyperbolic Adam algorithm [20]. We also use One Cycle learning rate schedule[30] for faster model convergence.

4. Experiments

In this section we provide the experimental results that compare the proposed Unsupervised Neural Quantization (UNQ) method with the existing unsupervised compression methods. Following the recent work[26], we perform the most of experiments on two sets of data:

1. **Deep1M/Deep10M/Deep1B** datasets contain 96-dimensional visual descriptors, which are computed from the activations of a deep neural network[3]. Base sets include 10^6, 10^7 and 10^9 vectors correspondingly. We use the additional separate sets of 500.000 vectors for training and 10.000 hold-out queries for evaluation.

2. **BigANN1M/BigANN10M/BigANN1B** datasets contain 128-dimensional histogram-based SIFT descriptors[15]. Base sets include 10^6, 10^7 and 10^9 vectors correspondingly. Here we also use the separate sets of 500.000 vectors for training and 10.000 hold-out queries for evaluation.

Unless stated otherwise, we always learn the method parameters on the train set, then compress the base set and evaluate the retrieval performance on the query set. As a common measure of compressed-domain retrieval performance we report *Recall@k* (for $k = 1, 10, 100$), which is the probability that the true nearest neighbor is among k closest neighbors in the compressed dataset. Two compression levels (8, 16 bytes per vector) were evaluated. In all the experiments we used the quantization codebooks of $K = 256$ codewords for all the methods.

Method	**BigANN1M**			**Deep1M**		
	R@1	R@10	R@100	R@1	R@10	R@100
	8 bytes per vector					
OPQ	20.8	64.3	95.3	15.9	51.3	88.6
Catalyst + OPQ	26.2	73.0	97.3	20.9	61.5	93.5
Catalyst + Lattice	28.9	75.8	97.9	24.6	68.3	96.1
LSQ	29.2	77.7	98.7	21.7	64.0	94.5
LSQ + rerank	30.3	78.9	98.8	22.8	65.7	95.6
UNQ	**34.6**	**82.8**	**99.0**	**26.7**	**72.6**	**97.3**
	16 bytes per vector					
OPQ	40.9	89.8	99.9	35.0	82.5	99.1
Catalyst + OPQ	46.1	92.0	99.8	39.0	86.5	99.3
Catalyst + Lattice	49.1	94.1	**100.0**	44.8	90.8	**99.8**
LSQ	57.1	97.5	**100.0**	41.1	88.6	99.5
LSQ + rerank	57.7	97.6	**100.0**	42.4	89.5	99.6
UNQ	**59.3**	**98.0**	**100.0**	**47.9**	**93.0**	**99.8**

Table 2. The compressed-domain retrieval performance achieved by unsupervised compression approaches. The proposed UNQ method outperforms all the competitors on both datasets and under both memory budgets.

4.1. Comparison to the state-of-the-art

As a preliminary experiment, we compare the proposed UNQ method with the current state-of-the-art approaches for the unsupervised compression problem on the million-scale **Deep1M** and **Bigann1M** datasets. In particular, we compare the following methods:

- **OPQ**[6, 24], with the implementation from the Faiss library[16].

- **Catalyst+OPQ**[26] that uses OPQ on top of the "spreaded" vectors, as described in [26]. We use the implementation provided by the authors and tune d_{out} and λ hyperparameters for optimal performance.

- **Catalyst+Lattice**[26] that uses the fixed predefined lattice on a sphere for vector quantization. Here we also use the implementation provided by the authors and tuned d_{out} and λ hyperparameters. The dimension of hidden layers was set to 2048 neurons. The lattice quantizers with $r^2 = 79$ for 8 bytes and $r^2 = 253$ for 16 bytes were used.

- **LSQ**[22], the state-of-the-art shallow quantization method that approximates each vector by a sum of codewords. We use the implementation provided by the authors.

- **LSQ+rerank**, that additionally reranks some top of LSQ results by the learned decoder with two hidden layers of 1024 neurons. The decoder obtains D-dimensional LSQ approximations as an input and is trained to minimize the reconstruction objective (9). The number of elements to rerank is the same as for UNQ.

- **UNQ**, our method, introduced in this paper. We use the architecture similar to [26]: the encoder and decoder consist of two 1024-unit linear layers, each equipped with Batch Normalization and ReLU activation function. The dimensionality of codewords was set to 256, and we rerank top-500 candidates.

The recall values achieved by the different methods are presented in Table 2. Below we highlight several key observations:

- On both datasets and for both compression levels the introduced Unsupervised Neural Quantization outperforms the competitors and provides a new state-of-the-art for the unsupervised compression problem.

- The current state-of-the-art methods **Catalyst+Lattice** and **LSQ** are competitive on different types of visual data. While **LSQ** outperforms **Catalyst+Lattice** on shallow SIFT descriptors, its accuracy is much lower on deep descriptors. Meanwhile, the proposed UNQ provides the highest accuracy on both datasets, which makes it a universal method for all types of data.

- An additional reranking stage with a learnable decoder provides only a slight improvement for the shallow **LSQ** method. This indicates that the end-to-end learning in **UNQ** is crucial for high compression accuracy.

Method	**BigANN10M**			**Deep10M**		
	R@1	R@10	R@100	R@1	R@10	R@100
	8 bytes per vector					
Catalyst + Lattice	20.9	63.9	94.3	18.2	53.5	88.7
LSQ	21.7	64.3	95.0	14.8	48.1	84.8
LSQ + rerank	21.8	64.8	95.0	15.1	48.8	85.9
UNQ	**25.8**	**70.5**	**96.3**	**18.8**	**57.0**	**90.9**
	16 bytes per vector					
Catalyst + Lattice	42.0	90.0	99.7	37.9	84.6	99.2
LSQ	50.5	95.0	**99.9**	34.3	79.8	98.2
LSQ + rerank	50.7	95.3	**99.9**	34.5	81.1	98.4
UNQ	**52.1**	**95.4**	**99.9**	**40.1**	**86.8**	**99.3**

Table 3. The performance on the ten million datasets. On this scale, the advantage of the proposed Unsupervised Neural Quantization method over existing approaches persists.

Method	**BigANN1B**			**Deep1B**		
	R@1	R@10	R@100	R@1	R@10	R@100
	8 bytes per vector					
Catalyst + Lattice	10.4	37.6	76.6	**16.8**	**38.7**	68.2
LSQ	9.6	35.9	73.3	13.2	32.3	59.9
LSQ + rerank	9.9	36.1	73.8	12.3	31.6	59.7
UNQ	**13.0**	**44.5**	**82.4**	14.5	37.8	**68.5**
	16 bytes per vector					
Catalyst + Lattice	31.1	77.8	98.3	35.3	72.8	95.6
LSQ	38.0	85.6	99.3	30.5	65.0	91.1
LSQ + rerank	37.6	86.0	99.3	30.1	65.8	91.4
UNQ	**38.3**	**86.8**	**99.4**	**35.5**	**74.2**	**96.1**

Table 4. The performance on the one billion datasets. The proposed UNQ method outperforms the competitors in most operating points.

4.2. Additional memory consumption

Here we analyze the additional memory consumption required by the proposed UNQ method. Compared to the shallow baselines UNQ additionally stores the parameters of the feed-forward encoder and decoder networks. In our experiments, the model requires about 19.8 Mb for 8-byte budget and 30.1 Mb for 16-byte budget, which is on par with the Catalyst+Lattice[26] which requires 17.2 Mb. Note, that this amount does not depend on the number of database vectors. For instance, for databases of one billion vectors, this results only in negligible 0.02 additional bytes per vector. As the most important experiment, we verify that the advantage of UNQ persists for more massive datasets. Namely, we perform the comparison of Catalyst+Lattice, LSQ, and UNQ on larger datasets Deep10M and BigANN10M, and billion-scale datasets Deep1B and BigANN1B. The results are shown in Table 3 and Table 4 and demonstrate that UNQ always outperforms a shallow LSQ counterpart by a considerable margin. UNQ also outperforms Catalyst+Lattice in most operating points, except for R@1,R@10 on 8-bytes encoding on Deep1B. For Deep1B and BigANN1B we rerank top-1000 candidates since it requires a negligible time in comparison with the billion-scale search.

4.3. Ablation

In this section we empirically validate our choice of training architecture and model by evaluating the contribution of each component. All experiments in this section fit the budget of M=8 bytes per vector on the BigANN1M dataset. More specifically, we compare

- **UNQ** — our primary model that is built and trained in accordance with the description provided in Section 3 with all parameters described in the first experiment.
- **Exhaustive reranking** — like **UNQ**, but the nearest

Method	BigANN1M, 8 bytes		
	R@1	R@10	R@100
UNQ	34.6	82.8	99.0
Exhaustive reranking	34.6	82.8	99.3
No reranking	25.0	68.5	95.0
No triplet loss	35.5	83.4	95.7
Triplet only	27.9	72.6	99.2
UNQ w/o hard	33.8	80.4	98.0
UNQ w/o Gumbel	30.2	75.7	78.1
No regularizer	31.0	80.4	95.2

Table 5. Ablation study for different training objectives.

neighbor search is performed with $d_1(\cdot,\cdot)$ only. This approach requires reconstructing every data vector by running the decoder module once for every vector in the database. This setup was evaluated on the same model parameters as **UNQ**.

- **No reranking** — the training procedure is the same as for **UNQ**, but the search is implemented without reranking by the decoder.
- **No triplet loss** — UNQ model that performs two-stage search but does not optimize for $d_2(\cdot,\cdot)$ explicitly, i.e. $\alpha = 0$. This is equivalent to training a regularized discrete variational autoencoder without explicit requirements to its hidden representation.
- **Triplet only** — like **UNQ**, but the nearest neighbor search is performed with $d_2(\cdot,\cdot)$ only. This model is also trained using $\alpha = 1.0$ and without the term (9).
- **UNQ w/o hard** — like **UNQ**, but Gumbel-Softmax without hard assignment discretization, as in [13].
- **UNQ w/o Gumbel** — like **UNQ**, but the quantization is implemented as in [1] with β=0.1.
- **No regularizer** — same as **UNQ**, but the balance regularizer term is set to $\beta = 0$. All other parameters are unchanged.

The results in Table 5 suggest that each of the three core components of our approach (reranking, triplet loss, CV regularizer) influences the model performance. The reranking stage with $L = 500$ candidates is predictably less important on R@100, compared to R@1, as one can see from the comparison between **UNQ** and **No reranking**. The triplet loss term benefits R@100, while the CV regularizer provides significant gains across all three recall areas. Note, that the usage of the Gumbel-Softmax trick outperforms the differentiable quantization, proposed in [1]. Finally, the usage of "hard" version of Gumbel-Softmax trick results in higher performance compared to **UNQ w/o hard** option.

4.4. Timings

Finally, we discuss the timings needed to encode the database and search over compressed data.

Encoding the database points with UNQ has almost the same complexity as in Catalyst as it requires the only feed-forward pass through two fully-connected layers. In particular, the encoding time of Deep1M for 8 bytes per point on the single Nvidia 1080Ti GPU for the UNQ requires about 1.5 seconds, while for the Catalyst+Lattice it is about 4.1 seconds on the same GPU card. The LSQ encoding is slower as it requires several optimization iterations, in our experiments it took 27 seconds to encode Deep1M with the authors' implementation.

Search. Unlike the existing competitors, the proposed UNQ method includes an additional reranking stage, that reconstructs a few candidates with the feed-forward decoder and computes the distances in the original D-dimensional space. Because the number of candidates is typically small, the reranking stage almost does not influence the total search runtime. E.g. on the Deep1B dataset with M=8 the exhaustive scan with $d_2(\cdot,\cdot)$ (via lookup tables) requires 3 seconds. Meanwhile, the reranking of 1000 candidates, implemented via BLAS instructions with the Intel MKL library, requires only 25.9 ms. Both timings are obtained in a single-CPU mode on the same machine. This indicates that the additional runtime cost from reranking is insignificant, especially for large databases or longer codes. Note, that the search in the Catalyst+Lattice method is slower compared to the LUT-based methods, namely, [26] reports about $1.5\times$ increase in search runtime for 8-byte codes.

5. Conclusion

We have presented Unsupervised Neural Quantization (UNQ) — a new unsupervised compression scheme for the problem of compressed-domain retrieval. Our scheme employs the ideas from the recent works on discrete autoencoders and shows that with the proper training objective the hidden variables can successfully serve as quantized representations for efficient retrieval. From another point of view, our method can be seen as a natural "deep" generalization of the existing shallow quantization methods, such as AQ or LSQ.

By a large number of experiments, we demonstrate the advantage of UNQ over the state-of-the-art approaches, such as LSQ and the recent lattice-based quantizer. Furthermore, while the existing methods perform differently on different types of data, UNQ provides the highest retrieval accuracy on both histogram-based and deep descriptors.

For the reproducibility purposes, we publish the Pytorch implementation of Unsupervised Neural Quantization online[2].

[2] https://github.com/stanis-morozov/unq

References

[1] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc V Gool. Soft-to-hard vector quantization for end-to-end learning compressible representations. In *Advances in Neural Information Processing Systems*, pages 1141–1151, 2017. 8

[2] Artem Babenko and Victor S. Lempitsky. Additive quantization for extreme vector compression. 2014. 1, 2

[3] Artem Babenko and Victor S. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. 2016. 5

[4] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIV*, pages 139–156, 2018. 3

[5] Yongjian Chen, Tao Guan, and Cheng Wang. Approximate nearest neighbor search by residual vector quantization. In *Sensors*, 2010. 2

[6] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. 2013. 1, 2, 5, 6

[7] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(12):2916–2929, 2013. 2

[8] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir D. Bourdev. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014. 1

[9] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. Spherical hashing. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 2957–2964, 2012. 2

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015. 4

[11] Himalaya Jain, Joaquin Zepeda, Patrick Pérez, and Rémi Gribonval. Subic: A supervised, structured binary code for image search. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 833–842, 2017. 1, 2

[12] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. 2016. 1, 3, 4

[13] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 8

[14] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011. 1, 2

[15] Herve Jegou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: Re-rank with source coding. In *Proc.ICASSP*, 2011. 5

[16] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017. 6

[17] Lukasz Kaiser, Samy Bengio, Aurko Roy, Ashish Vaswani, Niki Parmar, Jakob Uszkoreit, and Noam Shazeer. Fast decoding in sequence models using discrete latent variables. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 2395–2404, 2018. 3

[18] Benjamin Klein and Lior Wolf. In defense of product quantization. *CoRR*, abs/1711.08589, 2017. 1, 2

[19] Simon Lynen, Torsten Sattler, Michael Bosse, Joel A. Hesch, Marc Pollefeys, and Roland Siegwart. Get out of my lab: Large-scale, real-time visual-inertial localization. In *Robotics: Science and Systems XI, Sapienza University of Rome, Rome, Italy, July 13-17, 2015*, 2015. 1

[20] Jerry Ma and Denis Yarats. Quasi-hyperbolic momentum and adam for deep learning. *CoRR*, abs/1810.06801, 2018. 5

[21] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *CoRR*, abs/1611.00712, 2016. 1, 3, 4

[22] Julieta Martinez, Joris Clement, Holger H. Hoos, and James J. Little. Revisiting additive quantization. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II*, pages 137–153, 2016. 1, 2, 5, 6

[23] Julieta Martinez, Shobhit Zakhmi, Holger H. Hoos, and James J. Little. LSQ++: lower running time and higher recall in multi-codebook quantization. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*, pages 508–523, 2018. 1, 2

[24] Mohammad Norouzi and David J. Fleet. Cartesian k-means. 2013. 1, 2, 6

[25] Ezgi Can Ozan, Serkan Kiranyaz, and Moncef Gabbouj. Competitive quantization for approximate nearest neighbor search. *IEEE Trans. Knowl. Data Eng.*, 28(11):2884–2894, 2016. 1, 2

[26] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Hervé Jégou. Spreading vectors for similarity search. 2018. 2, 3, 5, 6, 7, 8

[27] Alexandre Sablayrolles, Matthijs Douze, Nicolas Usunier, and Herve Jegou. How should we evaluate supervised hashing? In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*, pages 1732–1736, 2017. 1

[28] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR*, abs/1701.06538, 2017. 5

[29] Raphael Shu and Hideki Nakayama. Compressing word embeddings via deep compositional code learning. *CoRR*, abs/1711.01068, 2017. 4

[30] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. *arXiv preprint arXiv:1708.07120*, 2017. 5

[31] George Tucker, Andriy Mnih, Chris J. Maddison, John Lawson, and Jascha Sohl-Dickstein. REBAR: low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 2624–2633, 2017. 3

[32] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6309–6318, 2017. 1, 3

[33] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 1753–1760, 2008. 1, 2

[34] Hanwei Wu and Markus Flierl. Learning product codebooks using vector quantized autoencoders for image retrieval. *CoRR*, abs/1807.04629, 2018. 3

[35] Tan Yu, Junsong Yuan, Chen Fang, and Hailin Jin. Product quantization network for fast image retrieval. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part I*, pages 191–206, 2018. 1, 2

[36] Jialiang Zhang and Jing Li. PQ-CNN: accelerating product quantized convolutional neural network on FPGA. In *26th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2018, Boulder, CO, USA, April 29 - May 1, 2018*, page 207, 2018. 1

[37] Ting Zhang, Chao Du, and Jingdong Wang. Composite quantization for approximate nearest neighbor search. 2014. 1, 2