Nearest Neighbour Search Using Binary Neural Networks

Demetrio Ferro
Polytechnic of Turin, Electronics and Telecommunications dpt.
Turin, Italy
ferro.demetrio@gmail.com

Vincent Gripon
Télécom Bretagne, Electronics dpt.
UMR CNRS Lab-STICC
Brest, France
vincent.gripon@telecom-bretagne.eu

Xiaoran Jiang
INRIA, Centre Rennes
Bretagne Atlantique
Rennes, France
xiaoran.jiang@inria.fr

***Abstract*—The problem of finding nearest neighbours in terms of Euclidean distance, Hamming distance or other distance metric is a very common operation in computer vision and pattern recognition. In order to accelerate the search for the nearest neighbour in large collection datasets, many methods rely on the coarse-fine approach.**

In this paper we propose to combine Product Quantization (PQ) and binary neural associative memories to perform the coarse search. Our motivation lies in the fact that neural network dimensions of the representation associated with a set of k vectors is independent of k. We run experiments on TEXMEX SIFT1M and MNIST databases and observe significant improvements in terms of complexity of the search compared to raw PQ.

I. Introduction

Nearest Neighbour Search is a common task required in different fields of application. Its definition is: "*Given a collection of data points and a query point in a multidimensional metric space, find the data point that is closest to the query point*" [1], where closeness of vectors is typically measured in terms of Euclidean distance, but still it applies with different distance metrics.

The geometrical solution consists of computing distances for all of the data points and retrieving the closest one. The issue involved with exhaustive Nearest Neighbour (NN) search is that its computational cost grows linearly with the cardinality and the dimensionality of the search space. Along the years, many techniques which aim to reduce the complexity have been proposed.

A possible approach is multi-dimensional indexing, aiming to build data structures optimized for queries such as KD-trees [2]. It was proven that it does not provide great enhancements when the search space dimensionality increases [3]. Another prominent way to reduce the search time is resorting to the Approximate Nearest Neighbour (ANN) techniques. The idea is to retrieve an approximate version of the exact NN, which matches true solution with fixed, high probability, not always unitary. One of the most widely adopted ANN techniques is Euclidean Locality Sensitive Hashing (E2LSH) [4]. Based on simple assumptions, it is able to achieve good performances. Relying on structured quantizers, it may not be sensitive enough to the distribution of real data [5]. The introduction of unstructured quantizers and clustering methods such as k-means leads to vector quantization techniques which perform much better since they are trained with real data [6]. However, using a lot of quantization cells over high dimension space may be a very challenging task both in terms of memory usage and computational cost.

Product Quantization (PQ) is a semi-structured quantization strategy which allows us to choose the number of vector space dimensions to be quantized jointly, in order to produce a large set of quantization cells from several small subsets [7]. Still it aims to preserve the quality of quantization, hence to reduce the Mean Square Error (MSE), in order to try working at good Rate-Distortion conditions [8]. Recently it was shown that it is possible to exploit the properties of Willshaw Neural Networks (WNNets) (binary associative memories) [9], [10] to accelerate the search over sparse, binary vector spaces [11].

The aim of this work is to try enhancing the PQ technique using clustered neural associative memories introduced recently [12], [13]. The main idea is to accelerate the query stage by selecting a subset of search space points which are likely to include the NN thanks to a bunch of associative memories based on neural networks. This approach is often referred to as coarse-fine search.

The outline of this paper is as follows. In Section II are introduced related works and notations, Section III presents our methodology, Section IV discusses our experiments and the conclusion is given in Section V.

II. Related works

For the purpose of accelerating PQ queries resorting to the associativity of binary neural networks, in this section is provided a general overview of the two main techniques reference and recalls to the tools used.

A. Exhaustive Search

Consider a set $\mathcal{X} \subset \mathbb{R}^d$ of N vectors $\mathbf{x}$ with dimension d. Given a query vector $\mathbf{x}_0 \in \mathbb{R}^d$, the problem of finding the NN consists of retrieving the closest $\mathbf{x} = \text{NN}(\mathbf{x}_0) \in \mathcal{X}$ in terms of Euclidean distance $d_E(\cdot)$:

$$\text{NN}(\mathbf{x}_0) := \underset{\mathbf{x}' \in \mathcal{X}}{\arg\min}\, d_E(\mathbf{x}', \mathbf{x}_0). \tag{1}$$

In the general case the complexity of the search is $\mathcal{O}(N \cdot d)$ and is tight [14]. That is why the predominant way to solve

NN by reducing complexity is to resort to ANN techniques. These techniques are twofold: a) they act on the number of elements N or b) they act on reducing dimension d.

B. Product Quantization

PQ [7] is a state-of-the-art method to solve ANN, acting as follows: the search space is split over m lower-dimensional orthogonal subsets, over which k Voronoi quantization regions are built. The d-dimensional Euclidean Distance computation is performed as the sum of m distances between the (d/m)-dimensional vectors. In the following subsections are explained the details of its principles.

1) Splitting and Quantization: Each vector $\mathbf{x} \in \mathcal{X}$ is split into m partitions, by means of a split operator $u_i : \mathbb{R}^d \rightarrow \mathbb{R}^{d/m}$, such that $\{u_1(\mathbf{x}), u_2(\mathbf{x}), \ldots, u_m(\mathbf{x})\}$ belong to orthogonal subspaces $\{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m\}$ of the search space $\mathcal{X}$.

For each of the possible $N \cdot m$ different vector splits $u_i(\mathbf{x})$, a quantization function $q_i(\cdot) : \mathcal{U}_i \rightarrow \mathcal{C}_i$ is applied, in order to map it into the closest centroid $\mathbf{c}_{i,j_i} \in \mathcal{C}_i$, where $\mathcal{C}_i$ is the i^{th} quantization grid obtained by k-means clustering.

The PQ stage consists of applying the quantization set of functions $q(\cdot) = (q_1(\cdot), q_2(\cdot), \ldots, q_m(\cdot))$:

$$[u_1(\mathbf{x}), \ldots, u_m(\mathbf{x})] \overset{q(\cdot)}{\rightarrow} [\mathbf{c}_{1,j_1}, \ldots, \mathbf{c}_{m,j_m}]. \tag{2}$$

The quality of the quantization, measured through the Mean Square Error (MSE), relies on the large cardinality of the centroids grid: $|\mathcal{C}| = |\mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_m| = k^m$. In other words, PQ performs quantization of vectors on the cartesian product of centroids.

2) Distance Computation: The PQ technique allows approximating Euclidean distance computation using the lower dimensional distances computed over search subspaces $\{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m\}$ at quantization stage. The d-dimensional distance is computed as the sum of m distances over (d/m)-dimensional vector partitions.

As introduced in [7], it is possible to adopt Symmetrical or Asymmetrical Distance Computation (SDC or ADC respectively). In both cases, the error introduced is upper-bounded by the quantization MSE. Since the distance distortion of the asymmetrical version is generally lower, in this work we focus on the case of ADC. Note that adapting our proposed solution to computing SDC is straight-forward.

The asymmetric approximation of Euclidean distance $\hat{d}_E(\cdot)$ is expressed as:

$$\hat{d}_E(\mathbf{x}, \mathbf{x}_0)^2 = \sum_{i=1}^{m} d_E(q_i(u_i(\mathbf{x})), u_i(\mathbf{x}_0))^2. \tag{3}$$

In order to accelerate the computation of distances, the values $d_E(q_i(u_i(\mathbf{x})), u_i(\mathbf{x}_0))$ can be stored in a $[k \times m]$ matrix to be used as a lookup table.

Solving the ANN search task by means of this approximation consists of finding:

$$\hat{\text{NN}}(\mathbf{x}_0) = \underset{\mathbf{x}' \in \bar{\mathcal{X}}}{\arg\min}\, \hat{d}_E(\mathbf{x}', \mathbf{x}_0). \tag{4}$$

whose complexity stands in quantization $\mathcal{O}(k \cdot d)$ and ADC $\mathcal{O}(m \cdot N)$, consider hence $\mathcal{O}(k \cdot d + m \cdot N)$.

In order to accelerate PQ search, we propose to use binary associative memories, described in the following section.

C. Binary Associative Memories

Willshaw Neural Networks are binary associative memories [9], [10] which can be used to measure how likely a query vector is compared to a collection of stored ones [11]. This lets us have a neuro-inspired approach aiming to solve the NN search. Recently, a new version of Willshaw neural networks have been proposed [12], [13] that achieve better performance by making use of the specific structure of stored vectors. It is this modification of Willshaw networks we consider in this approach.

Consider the set $\mathcal{Z} \subset \mathbb{Z}_2^{d'}$ of N vectors $\mathbf{z}$, to be the binary, sparse search space where we search for a d'-dimensional query vector $\mathbf{z}_0 \in \mathbb{Z}_2^{d'}$.

1) Learning Stage: In order to apply the associative memory model, let us partition the set $\mathcal{Z}$ into L subsets $\{\mathcal{Z}_1, \mathcal{Z}_2, \ldots, \mathcal{Z}_L\}$ completely disjoint, such that $\mathcal{Z} = \cup_{l=1}^{L} \mathcal{Z}_l$ and $\mathcal{Z}_l \cap \mathcal{Z}_{l'} = \emptyset$ for $l \neq l'$, with same cardinality $|\mathcal{Z}_l| = n$, hence $|\mathcal{Z}| = N = n \cdot L$. Note that this setting forces N to be a multiple of n, which is only required to ease the mathematical presentation of our proposed solution. If it is not the case, then the parts $\mathcal{Z}_l$ should be balanced to avoid large differences in their sizes.

Each of the L subsets trains a different Willshaw Network, whose connection matrix is defined as:

$$\mathbf{W}(\mathcal{Z}_l) = \max_{\mathbf{z} \in \mathcal{Z}_l} (\mathbf{z} \cdot \mathbf{z}^T). \tag{5}$$

2) Scores Computation: Once all of the networks have learnt the input database vectors, given a query vector $\mathbf{z}_0$, for each network $\mathbf{W}(\mathcal{Z}_l)$ is computed the score $s(\mathbf{z}_0, \mathcal{Z}_l)$, which measures the likelihood of the vectors stored in the l^{th} network to be NNs.

$$s(\mathbf{z}_0, \mathcal{Z}_l) = \mathbf{z}_0^T \cdot \mathbf{W}(\mathcal{Z}_l) \cdot \mathbf{z}_0. \tag{6}$$

Let $\{W_{i,j}^l\}_{i,j=1}^{d'}$ be the coefficients of the matrix $\mathbf{W}(\mathcal{Z}_l)$, $\{z_i\}_{i=1}^{d'}$ the elements of query vector $\mathbf{z}_0$. Since the adjacency matrix $\mathbf{W}(\mathcal{Z}_l)$ is always square and symmetrical, it is possible to consider upper triangular matrix product. The above quadratic form becomes:

$$s(\mathbf{z}_0, \mathcal{Z}_l) = \sum_{i=1}^{d'} z_i^2 \cdot W_{i,i}^l + 2 \sum_{\substack{i,j=1 \\ j>i}}^{d'} z_i \cdot z_j \cdot W_{i,j}^l. \tag{7}$$

As suggested in [11], the (z_i, z_j) coefficients product may be computed only for the non-zero terms $W_{i,j}^l \neq 0$. As a

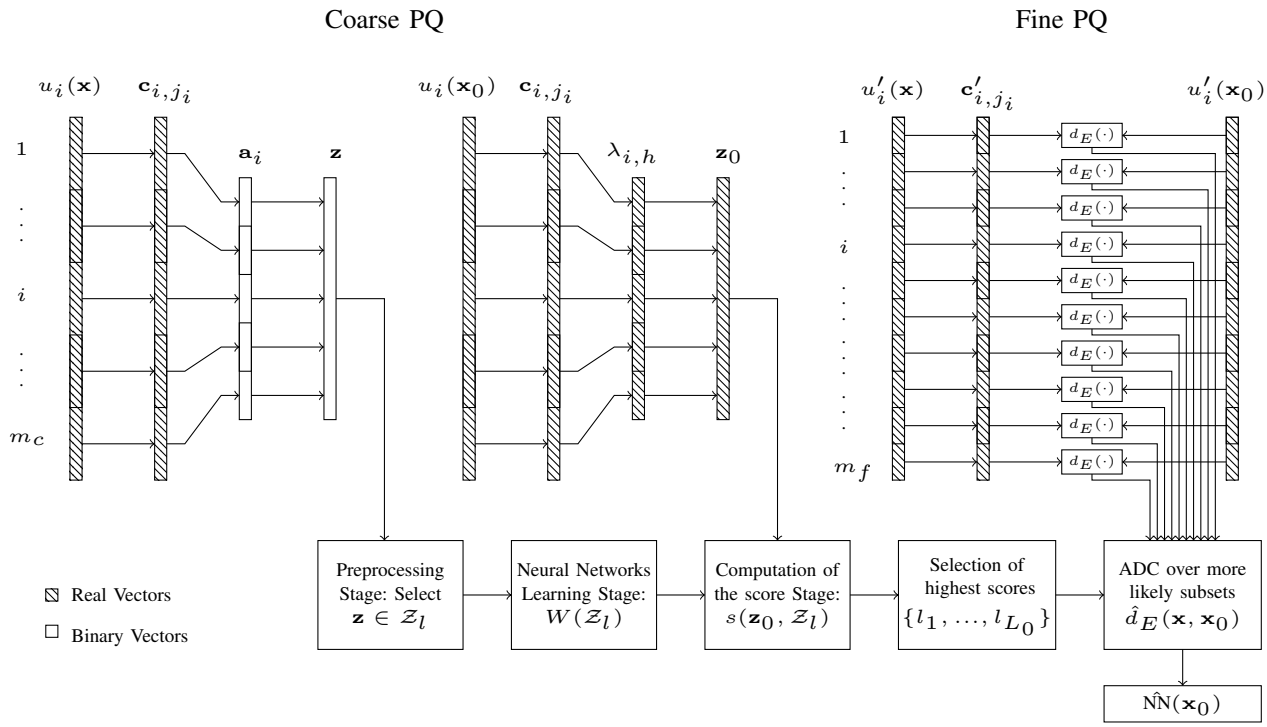

Fig. 1. Block scheme of the introduced technique.

result, computational cost of solving the above equation can be evaluated as $\mathcal{O}(p_1 \cdot d'^2)$, where p_1 is the mean value of unitary weights in the upper triangular matrix of all $\mathbf{W}(\mathcal{Z}_l)$ for $l \leq L$:

$$p_1 = \frac{1}{L} \frac{2}{d'(d'+1)} \sum_{l=1}^{L} \sum_{\substack{i,j=1 \\ j \geq i}}^{d'} W_{i,j}^{l}. \tag{8}$$

As a further observation, note that the score is maximum if $\mathbf{z}_0 \in \mathcal{Z}_l$, whereas it is low and close to zero if the query vector $\mathbf{z}_0 \notin \mathcal{Z}_l$. Note also that the score may be high even in this second case if the network is overfilled.

The selection of the search space points which are more likely to include the NN is then performed by picking the indices of the $L_0 \leq L$ networks which have highest scores of likelihood for the query vector.

The complexity of a query states in solving the quadratic form $\mathcal{O}(p_1 \cdot d'^2 \cdot L)$, and further exhaustive Euclidean Distance computation $\mathcal{O}(L_0 \cdot n \cdot d')$, in the overall consider it to be $\mathcal{O}(p_1 \cdot d'^2 \cdot L + L_0 \cdot n \cdot d')$.

III. Methodology

The technique shown here aims to accelerate PQ with clustered associative memories for NN search, by acting on the granularity of the quantization, with a coarse-fine approach.

In Figure 1 is reported an overall block scheme which shows the steps through the proposed ANN search solution. First, it is foreseen to run a *Coarse PQ*: the input data $\mathbf{x} \in \mathcal{X}$ are split over m_c orthogonal sections, over each of them is applied k-means to retrieve k_c centroids. In order to work with binary networks, each of the centroids is labelled by a sparse binary vector of dimension k_c, extracted from the canonical basis of $\mathbb{Z}_2^{k_c}$. Merging the labels associated with the centroids of each section implements a binary mapping of the search set quantization points. Once the search set is labelled, it is possible to train neural networks with binary vectors. At query time, the vector $\mathbf{x}_0$ is split and mapped into a vector of real value, containing similarity measures between the split of the query input and each of the possible k_c centroids. The score computation is then done for the similarity vector, under the assumption that the networks which are more likely to contain its NN achieve higher scores. Again it is possible to retrieve the $L_0 \cdot n$ more likely vectors, leading to a cardinality reduction of the search set.

Finally, the selected points are processed with a *Fine PQ* stage to run with more accuracy the ANN search. This implies a second split of the search set over m_f orthogonal sections, and the computation of k_f quantization points. Further details and explanations are reported in the following subsections.

A. Centroids Labelling

In the coarse PQ stage, it is built a grid of m_c splits and k_c centroids for each split. Consider the generic centroid $\mathbf{c}_{i,j_i} \in \mathcal{C}_i$, where $i \in [1, m_c], j_i \in [1, k_c]$. In order to work with binary networks, each vector partition $u_i(\mathbf{x}), \forall i \in [1, m_c], \forall \mathbf{x} \in \mathcal{X}$, is associated to a k_c-dimensional vector $\mathbf{a}_i$, extracted from the canonical binary basis $\mathcal{A} \subset \mathbb{Z}_2^{k_c}$ containing all zeros except

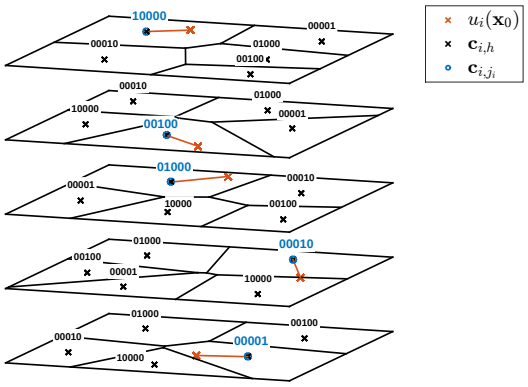

Fig. 2. Labelling rule $b(\cdot)$ for $d = 10, k_c = 5, m_c = 5$.

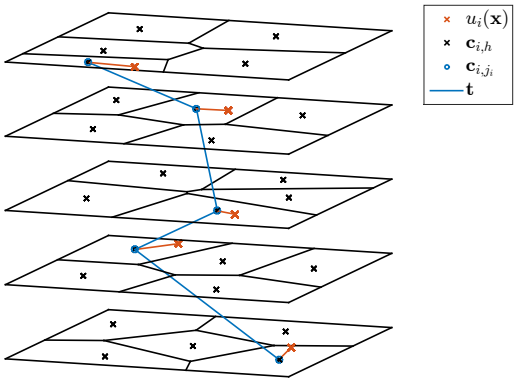

Fig. 3. Quantization point $\mathbf{t}$ for $d = 10, k_c = 5, m_c = 5$.

for the j_i-th position.

$$[\mathbf{c}_{1,j_1}, \mathbf{c}_{2,j_2}, \ldots, \mathbf{c}_{m_c,j_{m_c}}] \rightarrow [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{m_c}]. \tag{9}$$

e.g. if the centroid $\mathbf{c}_{1,j_1} = q_1(u_1(\mathbf{x}))$, associated to the first split of the quantized vector $u_1(\mathbf{x})$, is the j_1-th among the k_c set by coarse k-means clustering, it will be labelled as:

$$\begin{array}{ccccccccccc} & & & 1 & 2 & \ldots & j_1 & \ldots & k_c{-}1 & k_c & \\ \mathbf{a}_1 & = & [0, & 0, & \ldots, & 1, & \ldots, & 0, & 0]. \end{array} \tag{10}$$

Applying this labelling rule to all of the m_c space partitions, it is possible to build the vectors $\mathbf{z} \in \mathcal{Z} \subset \mathbb{Z}_2^{d_c}$, defined by the concatenation of all the $\mathbf{a}_i, i = 1, \ldots, m_c$, so that $d_c = k_c \cdot m_c$.

$$\mathbf{z} \quad = \quad [\quad \mathbf{a}_1 \quad || \quad \mathbf{a}_2 \quad || \quad \ldots \quad || \quad \mathbf{a}_{m_c} \quad]. \tag{11}$$

By defining $b : \mathbb{R}^d \rightarrow \mathbb{Z}_2^{d_c}$ as the labelling rule performing the just introduced criteria, consider hence $\mathbf{z} = b(\mathbf{x})$.

As reference, Figure 2 is a graphical representation of labelling rule application, relative to a simplified case.

The binary vectors $\mathbf{z} \in \mathcal{Z}$ obtained in this way, are used to train the Willshaw type associative memories, by applying the learning rule defined in subsection II-C1.

B. Preprocessing for Allocation

The selection of the vectors to allocate in each neural network has direct impact on the ration of ones in the connection matrices, considered here by the factor p_1.

For the specific structure of the training vectors, the value of p_1 can be bounded as:

$$\frac{m_c(m_c+1)}{d_c(d_c+1)} \leq p_1 \leq 1 - \frac{k_c - 1}{d_c + 1}. \tag{12}$$

The lower-bound is reached when all of the stored vectors are identical, (i.e. their mutual Hamming distance is zero), and the upper-bound is reached when all of the possible different vectors are stored (i.e. their mutual Hamming distance is maximal and equals $2 \cdot m_c$). In the first case, the number of 1s in the upper triangular matrix equals $m_c(m_c+1)/2$, whereas in the second case it is $d_c + k_c^2 m_c(m_c-1)/2$.

Let us suppose that the l^{th} learning set is composed of n binary label vectors $\mathcal{Z}_l := \{\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(n)}\}$.

For the specific structure of the label vectors, it is possible to derive a tighter approximation for p_1 as a function of n:

$$p_1(n) \approx \frac{2 \cdot \alpha(n) + m_c(m_c-1)k_c^2 \cdot \beta(n)}{d_c(d_c+1)}. \tag{13}$$

where $\alpha(n)$ and $\beta(n)$ come from the following considerations:

- $\alpha(n)$ is related to the average number of learnt vectors associated to different quantization points $\mathbf{c} \in \mathcal{C}_i$ within each search space partition $\mathcal{U}_i, i \in [1, m_c]$, i.e. the average Hamming weight $w_H(\cdot)$ of the OR operation applied to all of the learnt labels;

$$\alpha(n) = w_H(\mathbf{z}^{(1)} \vee \mathbf{z}^{(2)} \vee \ldots \vee \mathbf{z}^{(n)}) \tag{14}$$

- $\beta(n)$ is an approximation obtained as the average number of connections in the upper triangular side of the matrix $\mathbf{W}(\mathcal{Z}_l)$, trained by n independent and uniformly distributed vectors, [13].

$$\beta(n) = 1 - \left(1 - 1/k_c^2\right)^n. \tag{15}$$

For the lower-bound, consider $\alpha(n) = m_c$, $\beta(n) = 1/k_c^2$, (or, equivalently, set $n = 1$), whereas for the upper-bound consider $\alpha(n) = d_c$, $\beta(n) = 1$, (or let $n \rightarrow \infty$).

For any fixed n such that $n \ll k_c^2$, the factor $\beta(n)$ can be approximated as $\beta(n) \approx n/k_c^2$. In this case, since both $\alpha(n)$ and $\beta(n)$ grow linearly with n, it is suggested to use many networks trained by a small number of vectors.

Both in terms of performance and computational complexity, a good allocation strategy should try to keep the value of p_1 as low as possible. It is hence suggested to use an allocation strategy which tries to keep the lowest possible

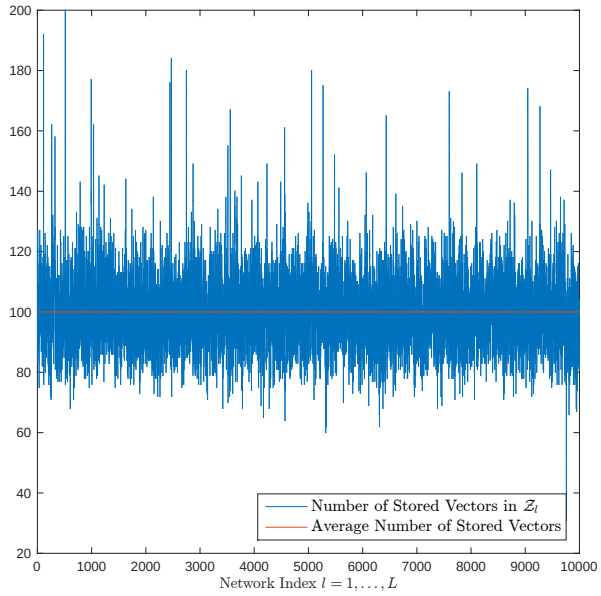

Fig. 4. Number of stored Vectors in each memory.

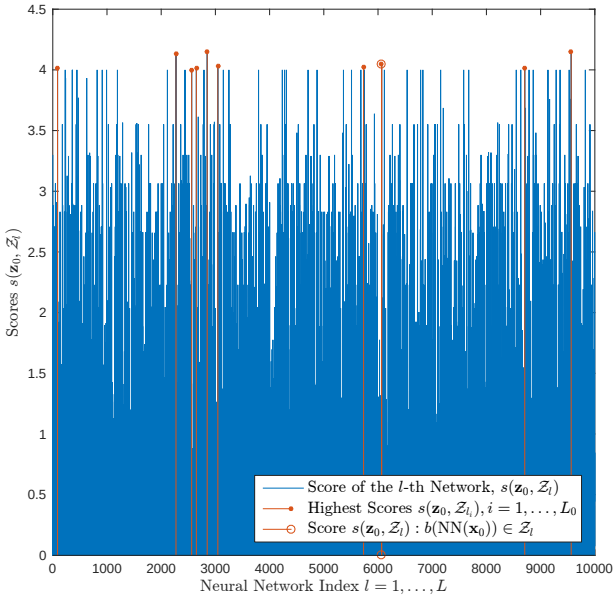

Fig. 5. Scores of each memory to include the $\mathrm{NN}(\mathbf{x}_0)$.

mutual Hamming distance label vectors.

The strategy used here has a greedy approach which tries to pick L different vectors, from here on named *first training vectors*, to be learnt by each of the associative memories. Then all of the $N - L = (n-1)L$ leaving data points are stored in the memory whose first training vector has minimum Hamming distance.

With the idea of being sensitive to the data distribution, the selection of the first training vectors takes into account the set of all of the unique quantization points $\mathbf{t} \in \mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_{m_c}$, to which data points are associated. Figure 3 reports a simple exemplification of one of the possible points $\mathbf{t}$ set by a Coarse PQ stage. First of all, set the index $l_{\mathbf{t}} = 0$, that is the index of the learning set to be populated by data points quantized as $\mathbf{t}$. Then, for each of the unique quantization points, it is supposed to execute the following steps:

- Compute $n_{\mathbf{t}} = |\mathcal{X}_{\mathbf{t}}|$, where $\mathcal{X}_{\mathbf{t}}$ is defined as: $\mathcal{X}_{\mathbf{t}} := \{\mathbf{x} \in \mathcal{X} : q([u_1(\mathbf{x}), u_2(\mathbf{x}), \ldots, u_{m_c}(\mathbf{x})]) = \mathbf{t}\}$, the set of (undistinguishable) data points quantized as $\mathbf{t}$.
- Choose randomly, a number $\nu_{\mathbf{t}} = \lfloor n_{\mathbf{t}} \cdot L/N \rfloor$ of vectors $\mathbf{x}$ in the set $\mathcal{X}_{\mathbf{t}}$, to initialize progressively the learning sets $\mathcal{Z}_l, \forall l \in [l_{\mathbf{t}} + 1, l_{\mathbf{t}} + \nu_{\mathbf{t}}]$.
- Update the index $l_{\mathbf{t}} = l_{\mathbf{t}} + \nu_{\mathbf{t}}$.

At the end of the iteration, since $\sum_{\mathbf{t} \in \mathcal{C}} \nu_{\mathbf{t}} \leq L$, the number of indices chosen may not be large enough, it is performed an extra random choice among the not chosen data points.

As shown in Figure 4, running this technique over $N = 10^6$ vectors, the number of stored vectors in each of the $L = 10^4$ networks is fairly distributed with mean $n = 10^2$.

C. Similarity Measure

At run time, the query vector undergoes coarse PQ, to be mapped into a set of centroids. This time, instead of labelling the centroids in a binary fashion, let us try to find a soft rule which is more sensitive to the distance among quantization points. Consider the generic centroid $\mathbf{c}_{i,h} \in \mathcal{C}_i$, and the minimum distance one $q_i(u_i(\mathbf{x}_0)) = \mathbf{c}_{i,j_i} \in \mathcal{C}_i$ where $i \in [1, m_c], h, j_i \in [1, k_c]$.

Let us define the value $\lambda_{i,h} \in [0, 1] \subset \mathbb{R}$ as follows:

$$\lambda_{i,h}(\mathbf{x}_0) = \frac{d_E(\mathbf{c}_{i,j_i}, u_i(\mathbf{x}_0))^2}{d_E(\mathbf{c}_{i,h}, u_i(\mathbf{x}_0))^2}, \tag{16}$$

which measures similarity for the query vector partition to be quantized in one of the k_c possible centroids. Note that instead of using $d_E(\cdot)$, one may choose a different metric depending on the application. It is applied to each of the centroids and to each of space partitions to build $\mathbf{z}_0 \in \mathbb{R}^{d_c}$:

$$\begin{aligned} \mathbf{z}_0 = \quad & [\lambda_{1,1}(\mathbf{x}_0), \quad \ldots, \quad \lambda_{1,k_c}(\mathbf{x}_0)|| \\ & \ldots \\ & ||\lambda_{m_c,1}(\mathbf{x}_0), \quad \ldots, \quad \lambda_{m_c,k_c}(\mathbf{x}_0)], \end{aligned} \tag{17}$$

which is a soft version of likelihood scores computation reported in section II-C2.

In Figure 5 it is possible to see how the score computation relies a high score when the tested memory stores data points which are very close to the $\mathrm{NN}(\mathbf{x}_0)$, whereas the score is lower when the data points are further.

D. Retrieving the solution

After the computation of the scores $s(\mathbf{z}_0, \mathcal{Z}_l), \forall l = 1, \ldots, L$, one may select the $L_0 \leq L$ highest peaks. In Figure

5 are reported the scores calculated for a certain query vector. Corresponding simulation parameters are $L = 10^4$, $n = 10^2$, $k_c = 32$ and $m_c = 2$. The $L_0 = 10$ highest peaks illustrated in red are supposed to contain the exact NN.

Consider $l_1, l_2, \ldots, l_{L_0}$, as the indices corresponding to the highest scores. First, retrieve the learning sets $\{\mathcal{Z}_{l_1}, \mathcal{Z}_{l_2}, \ldots, \mathcal{Z}_{l_{L_0}}\}$ which contain the most likely partitions of $\mathcal{Z}$. By doing this, it is possible to get back the most likely set of vectors $\mathbf{x} \in \mathcal{X}$ over which fine ADC computation with the vector $\mathbf{x}_0$ is run. Let us denote $\mathcal{X}_0 = \cup_{i=1}^{L_0} \mathcal{X}_{l_i}$, where $\mathcal{X}_{l_i}$ are the the equivalent partitions matching the index partitioning of $\mathcal{Z}$ defined in subsection II-C1. This leads to the solution of the NN search over a subset of the search space with lower cardinality, since $|\mathcal{X}_0| = L_0 \cdot n \leq N$.

The complexity of a query has to take into account two quantization layers $\mathcal{O}((k_c + k_f) \cdot d)$, scores computation $\mathcal{O}(p_1 \cdot d_c^2 \cdot L)$ and ADC over $\mathcal{X}_0$, $\mathcal{O}(L_0 \cdot n \cdot m_f)$. Consider hence an overall cost of $\mathcal{O}((k_c+k_f)\cdot d+p_1\cdot d_c^2\cdot L+L_0\cdot n\cdot m_f)$.

Compared to the cost of using just a Fine PQ technique, it is possible to note that the cardinality reduction $|\mathcal{X}_0| \leq |\mathcal{X}|$ has a direct impact on the computational cost, provided that coarse quantization and scores computation cost is much smaller than the Fine PQ one:

$$k_c \cdot d + p_1 \cdot d_c^2 \cdot L + L_0 \cdot n \cdot m_f \ll L \cdot n \cdot m_f. \qquad (18)$$

IV. Experiments

The method was evaluated both for retrieving the exact NN in a set of recalls and for classification. NN search was performed over TEXMEX SIFT1M ($N = 10^6$) local descriptors dataset from IRISA [15], whereas NN search for classification was performed over MNIST ($N = 60000$) handwritten digits dataset from NIST [16].

In Figure 6 are reported Performances vs Computational Cost scaled to Fine PQ (exhaustive PQ search) Cost for NN search. Over the abscissa, it is reported the overall computational cost Normalized to the one of running just Fine PQ:

$$\frac{k_c \cdot d + p_1 \cdot d_c^2 \cdot L + L_0 \cdot n \cdot m_f}{L \cdot n \cdot m_f}. \qquad (19)$$

In this first case, it is possible to note that the technique is able to provide complexity reduction up to 10 times, provided by search space cardinality reduction ($L_0 = L/10$), without affecting the performance of Fine PQ applied to the whole data set, reported as the asymptotic trend of the performance on the right of the plot, known from the state-of-the-art.

In Figure 7 are reported Performances vs Computational Cost scaled to Fine PQ Cost for Classification. Here is possible to notice that the performance does not grow monotonically and that it assumes high values even when the cardinality reduction is very large ($|\mathcal{X}_0| \ll |\mathcal{X}|$). Besides this, note that in this case scores computation have a higher impact on the overall cost.

In the above figures it is possible to appreciate how, compared both to Exhaustive NN search and raw PQ (with parameters k_f, m_f), the proposed technique represents a good trade-off between complexity reduction and accuracy of the solution.

V. Conclusions

We proposed a method to accelerate approximate nearest neighbour search by combining Product Quantization and binary neural associative memories for the coarse step. Our simulations on datasets TEXMEX SIFT1M and MNIST show that this neuro-inspired approach significantly outperforms the traditional PQ in terms of computational complexity with a limited impact on accuracy.

At first sight it seems complicated to merge binary neural networks with nearest neighbour search as the latter typically needs distance metrics to be computed. Coarse PQ acts as a perfect proxy here, as it projects real valued vectors to centroids which can be likened to values in a discrete alphabet. As a further observation, note that once the number L of networks to train is set, the choice of the parameters for coarse PQ is crucial. Whilst they have a linear impact on the size of the memories, they let cardinality of all possible messages to store grow with $k_c^{m_c}$.

Future work includes a) getting rid of the fine search stage, b) cascading (hierarchical) associative memories for better performance, c) proposing refined allocation strategies for a better choice of learning sets and d) performing simulations on harder datasets (e.g. GIST1M, SIFT1B).

Acknowledgments

This work has been funded in part by the European Research Council under the European Union's Seventh Framework Programme (FP7 / 2007 - 2013) / ERC grant agreement n 290901 NEUCOD.

References

[1] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In Catriel Beeri and Peter Buneman, editors, *Database Theory - ICDT'99*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer Berlin Heidelberg, 1999.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. In *Communications of the ACM*, pages 509–517, 1975.

[3] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the International Conference on Very Large DataBases*, pages 194–205, 1998.

[4] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, January 2008.

[5] Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. Locality sensitive hashing: a comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358, August 2010.

[6] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2161–2168, Washington, DC, USA, 2006. IEEE Computer Society.

[7] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, January 2011.

[8] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Searching with quantization: approximate nearest neighbor search using short codes and distance estimators. Research Report RR-7020, August 2009.

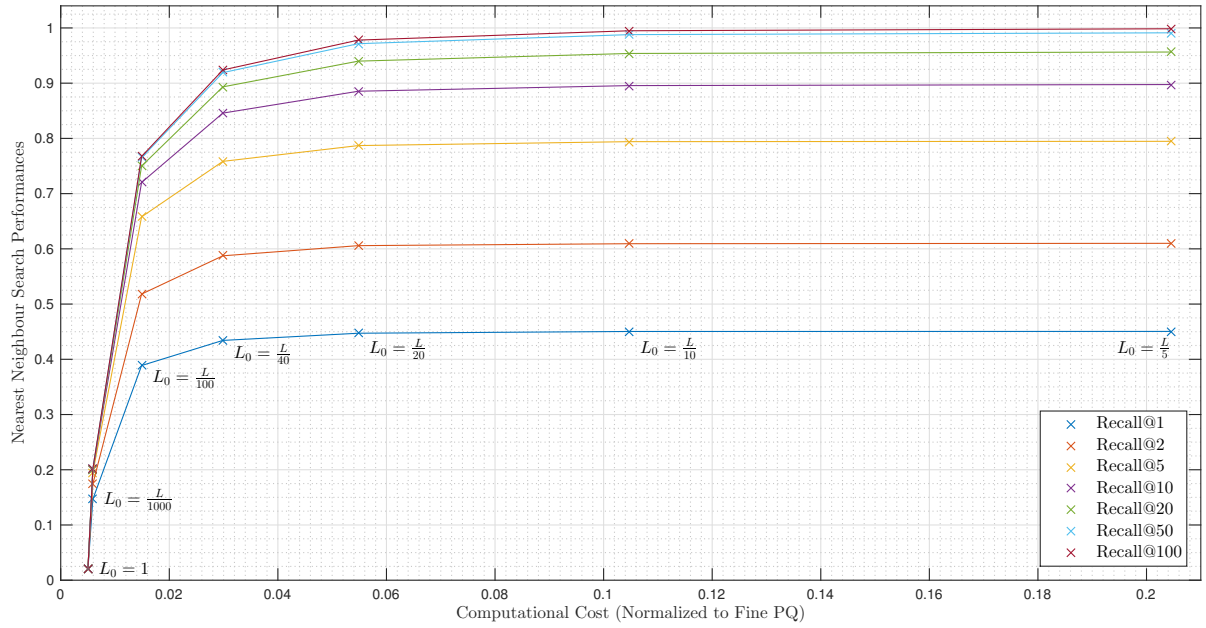

Fig. 6. NN Search, TEXMEX SIFT Descriptors.

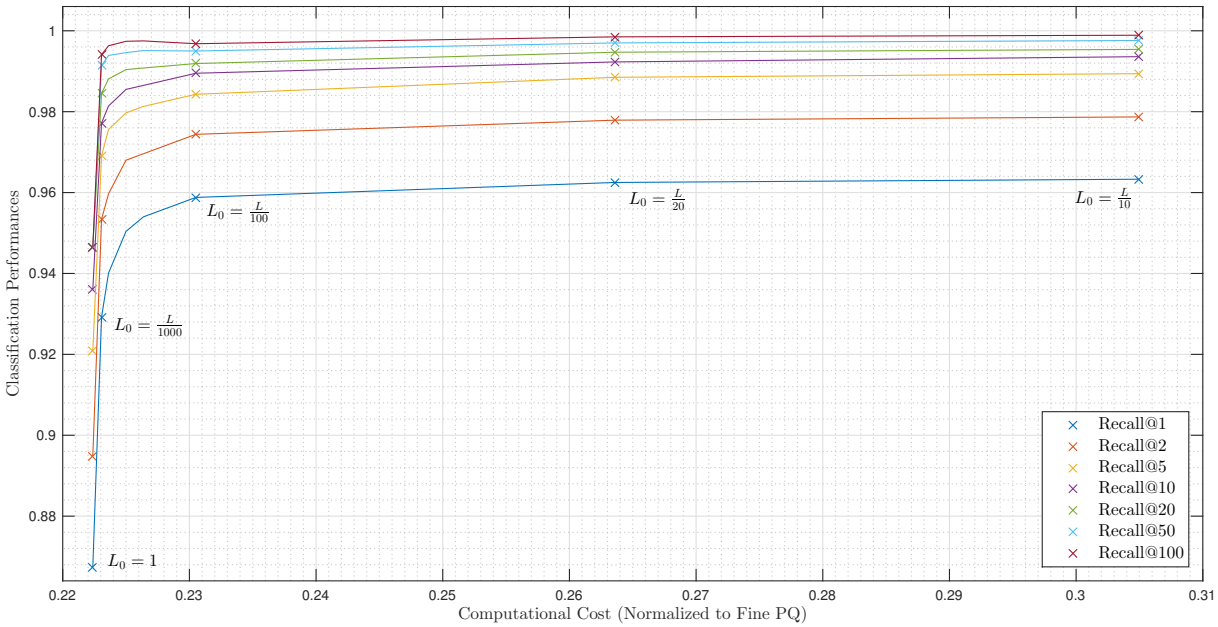

Fig. 7. Classification, MNIST handwritten digits.

[9] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins. Non-holographic associative memory. *Nature*, 222:960–962, June 1969.

[10] Richard N. A. Henson and David J. Willshaw. Short-term associative memory. *Proc INNS World Congress on Neural Networks*, 1995.

[11] Chendi Yu, Vincent Gripon, Xiaoran Jiang, and Hervé Jégou. Neural associative memories as accelerators for binary vector search. In *Proceedings of Cognitive*, March 2015. To appear.

[12] Vincent Gripon and Claude Berrou. A simple and efficient way to store many messages using neural cliques. In *Proceedings of IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain*, pages 54–58, Paris, France, April 2011.

[13] Vincent Gripon and Claude Berrou. Sparse neural networks with large learning diversity. *IEEE Transactions on Neural Networks*, 22(7):1087–1096, July 2011.

[14] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[15] Laurent Amsaleg and Hervé Jégou. Texmex - datasets for approximate nearest neighbor search, July 2010.

[16] Yann LeCun and Corinna Cortes. Mnist - database of handwritten digits.