In Defense of Product Quantization

Benjamin Klein*[1] and Lior Wolf†[1,2]

[1]The Blavatnik School of Computer Science, Tel Aviv University, Israel
[2]Facebook AI Research

Abstract

Despite their widespread adoption, Product Quantization techniques were recently shown to be inferior to other hashing techniques. In this work, we present an improved Deep Product Quantization (DPQ) technique that leads to more accurate retrieval and classification than the latest state of the art methods, while having similar computational complexity and memory footprint as the Product Quantization method. To our knowledge, this is the first work to introduce a representation that is inspired by Product Quantization and which is learned end-to-end, and thus benefits from the supervised signal. DPQ explicitly learns soft and hard representations to enable an efficient and accurate asymmetric search, by using a straight-through estimator. A novel loss function, Joint Central Loss*, is introduced, which both improves the retrieval performance, and decreases the discrepancy between the soft and the hard representations. Finally, by using a normalization technique, we improve the results for cross-domain category retrieval.*

1. Introduction

Computer vision practitioners have adopted Product Quantization (PQ) methods as a leading approach for conducting Approximated Nearest Neighbor (ANN) search in large scale databases. However, recently, the research community has shifted its focus toward using Hamming Distance on binary representations learned by a supervised signal, and showed its superiority on the standard PQ techniques [7]. In this work, we present a technique inspired by PQ and named Deep Product Quantization (DPQ) that outperforms previous methods on several common benchmarks in the field. While standard PQ is learned in an unsupervised manner, our DPQ is learned in an end-to-end fashion, and benefits from the task-related supervised signal.

PQ methods decompose the embedding manifold into a Cartesian product of M disjoint partitions, and quantize each partition into K clusters. An input vector $x \in \mathbb{R}^{MD}$ is decomposed into M sub-vectors in $\mathbb{R}^D$, $x = [x_1, x_2, \dots, x_M]$ and then encoded by PQ as $z_x \in \{0,1\}^{M \cdot \log_2(K)}$. Each group of $\log_2(K)$ bits decodes the index $k \in \{1 \dots K\}$ of the cluster to which the sub-vector belongs (the clusters vary between the subspaces). A representative vector $c_{m,k} \in \mathbb{R}^D$ is associated with each cluster k of each partition m. An approximation of the original vector, $\tilde{x}$, can be readily reconstructed from z_x by concatenating the representative vectors of the matching clusters. The common practice for training a PQ is to run K-means in an unsupervised manner on each partition, and to use the centroid of each resulting cluster as the representative vector. This approach focuses only on minimizing the distance between the original vector, x, and the compressed vector, $\tilde{x}$, and does not benefit from the task-related supervised signal, when such a signal is available.

The advantages of using the PQ technique are the reduction in memory footprint and acceleration of search time. The decomposition of the embedding into a Cartesian product of M sub-vectors is the key ingredient in the effectiveness of PQ in reducing the retrieval search time, since it allows to compute the approximated distance of a pair of vectors x and y directly from their compressed representation, z_x and z_y, using look-up tables. PQ methods can also achieve better retrieval performance by using an asymmetric search in which the distance is computed between the source vector, x, and the compressed vector, z_y, in the same amount of computation as the symmetric search. The ability of PQ to reduce the memory footprint and the retrieval search time, while preserving the search quality, were the major contributors to its success and continuing popularity.

Another common technique for ANN search is transforming the embedding into a binary representation and performing the comparison using hamming distance. Several works have achieved state of the art results on retrieval benchmarks by learning the binary representation as part

*beni.klein@gmail.com
†liorwolf@gmail.com

as of the classification model optimization in an end-to-end fashion. The binary representation is thus trained using a supervised signal and, therefore, the hamming distance between two binary representations reflects the end goal of the system.

In this work, we present a new technique, Deep Product Quantization, which to our knowledge, is the first to learn a compressed representation inspired by PQ which is learned end-to-end and, therefore, benefits from the supervised signal. Our contribution includes: (i) An end-to-end PQ approach for ANN search that exploits high-dimensional Euclidean distances instead of the hamming distance. (ii) Learning soft and hard representations as part of the training to facilitate symmetric and asymmetric search. (iii) Using a straight-through estimator to overcome the non-differential argmax function which is essential for our hard representation. (iv) A new loss function named *joint central loss*, which is inspired by the center loss [17] but also decreases the discrepancy between the soft and the hard representations. (v) A normalization technique which improves the results for cross-domain category retrieval.

2. Related work

Vector quantization techniques have been used successfully in the past in many applications, including data compression, approximated nearest neighbor search, and clustering. The most classic technique is Vector Quantization (VQ) which divides the space into K clusters, by using an unsupervised clustering method such as K-means. VQ allows to encode each sample by $\log_2(K)$ bits, namely by encoding the identity of the cluster to which the sample belongs. By precomputing the euclidean distance between every two clusters and storing the results in a hash table with $O(K^2)$ entries, one can compute the approximated distance between every two samples in $O(1)$ time. Since the number of clusters grows exponentially with the number of bits, one may expect the performance of VQ to improve as more bits are added. In practice, since VQ is learned using the K-means algorithm, a meaningful quantization of the space requires a number of samples which is proportional to the number of clusters. Additionally, since the hash table grows quadratically in the number of clusters, it becomes infeasible to use the hash table for large values of K. These reasons have limited the efficient usage of VQ to a small number of clusters. This limitation has an impact on the quantization error, i.e., the distance between the original vector and its matching centroid and, therefore, is a bottleneck in decreasing the quantization error and in improving the retrieval performance.

Product Quantization [8] (PQ) is a clever technique to unlock the bottleneck of increasing the number of clusters with respect to VQ, while allowing an efficient computation of the approximated euclidean distance between two compressed representations, and reducing the quantization error. The main idea is to divide a space in $\mathbb{R}^{MD}$ to a Cartesian product of M sub-vectors in $\mathbb{R}^D$. The VQ technique is then applied on each group of sub-vectors, resulting in M solutions of K-means in $\mathbb{R}^D$, where each solution has a different set of K clusters. Each vector in $\mathbb{R}^{MD}$ can be encoded using $M \cdot \log_2(K)$ bits, by assigning the index of the matching cluster to each of its M sub-vectors. The expressive power of PQ empowers it to transform a vector in $\mathbb{R}^{MD}$ to one of $K^M = 2^{M \cdot \log_2(K)}$ possible vectors, thus creating an exponential number of possible clusters in the number of available bits.

As discussed in Sec. 3.1, the PQ technique enables an efficient computation of the approximated distance between two compressed vectors using $O(M)$ additions. This is achieved by using M Look Up Tables (LUT) that store the distance between every two clusters for each of the M partitions. Additionally, the K-means algorithm is not bounded by the number of samples, since each run clusters the partitioned space to K clusters, where K is usually small (e.g. $K = 256$). Decreasing the quantization error even further, the PQ technique is also able to efficiently compare an uncompressed query vector to a database of compressed vectors. The latter is called asymmetric search, and the former is called symmetric search. Asymmetric search is the common practice in information retrieval systems that employ PQ, since while the database vectors need to be compressed in order to reduce their memory footprint, there is usually no memory limitation for the query, which typically arrives on-the-fly. In PQ, the asymmetric search has been shown to have a lower quantization error, while maintaining the computational complexity of the symmetric search by constructing Look Up Tables for each query.

The PQ technique has been widely adopted by the information retrieval and computer vision community and has started a long list of improvements to the original PQ technique; Optimized Product Quantization [6] (OPQ) and Cartesian K-means [15] have focused on improving the space decomposition and the learning of the optimal codebooks for decreasing the quantization error. These contributions rely on the observation that simply dividing the features to a Cartesian product does not fully utilize the knowledge about the structure of the feature space, and ignores the intra-subspace correlations of the data. To create a better partition of the space, they suggest to first transform the data by an orthonormal matrix, R, and then to do the Cartesian decomposition and learn the optimal clusters. LOPQ [10] used the observation that while PQ and OPQ create an exponential number of possible centroids in $\mathbb{R}^{MD}$, many of them remain without data support, and therefore are not efficiently used. To mitigate this problem, they suggest to first use a coarse quantizer to cluster the data, and capture its density, and then apply a locally optimized product quantization to each coarse cell.

Despite their tremendous success, Product Quantization techniques and Vector Quantization techniques in general, are being optimized in an unsupervised manner, with the goal of reducing the quantization error. While the quantization error is often correlated with the end task, the performance can be further improved by incorporating a supervised signal. The first contributions to incorporate such supervision employed a Hamming Distance on binary representations, which is a popular alternative technique for ANN.

Given two vectors, which are both encoded by $M \cdot \log_2(K)$ bits, the possible number of different distance values between them under hamming distance is only $M \cdot \log_2(K) + 1$. In contrast, the possible number of different distance values between them using PQ is $\binom{K}{2}^M$, which is much higher than hamming. The richness of the expressive power of PQ has allowed it to outperform hamming distance techniques that were trained in an unsupervised manner. With the advent of Deep Learning, many binary encoding techniques [18, 12, 13, 7] that utilize end-to-end training and, therefore, benefit from the supervised signal, have been suggested and have proven to be better than the standard PQ technique that is trained in an unsupervised manner [7].

Our work combines the expressive power of the PQ technique with Deep Learning end-to-end optimization techniques, and allows for PQ to benefit from the task related supervised signal. To our knowledge, we are the first to incorporate a technique inspired by PQ into a deep learning framework. Another work [3] has proposed to combine PQ with Deep Learning for hashing purposes, but in contrast to our work, they do not optimize the clusters of PQ with respect to the supervised signal of classification or retrieval. Instead, they alternate between learning PQ centroids using K-means on the embeddings space in an unsupervised fashion, and between learning the embedding using a CNN. Our solution learns the centroids and the parameters of the CNN end-to-end while optimizing the centroids explicitly to perform well on classification and retrieval tasks. We show in Tab. 1, that our technique is able to improve their results.

While our technique is inspired by Product Quantization, there are a few important technical distinctions; While in PQ the soft representation which is used for asymmetric search is the embedding itself and it is not constrained by the vectors of the clusters, in our work as described in Sec. 3, the soft representation is learned, and it is the concatenation of M soft sub-vectors, where each soft sub-vector is a convex combination of the learned centroids. Additionally, while the asymmetric search capability of PQ improves its performance, it is not explicitly optimized for, and its success is an outcome of the method's design. In contrast, our method learns both the soft and hard representations as part of the training, and directly improves the asymmetric search. This is done by using our innovative loss function, the *joint central loss* which is inspired by the *center loss* work [17]. The center loss aims to improve the retrieval performance of a CNN by learning a center for each class, and adds a term that encourages the embeddings to concentrate around the center of their corresponding class. Our joint central loss is adding another role to center loss, which is to decrease the discrepancy between the soft and the hard representations. This is achieved by optimizing both representations to concentrate around the same class centers.

Recently, a structured binary embedding method called SUBIC was proposed [7]. In their work, which is the current state of the art for retrieval, each sample is represented by a binary vector of MK bits, where in each group of K bits, only one bit is active. Therefore, each sample can be encoded by $M \cdot \log_2(K)$ bits. Similar to other works, the binary representation of SUBIC is not learned explicitly. Instead, each group of K entries is the result of the softmax function, and therefore acts as a discrete distribution function on $\{1, \dots, K\}$. In the inference phase, the entry that corresponds to the highest probability is taken to be the active bit, and all the others are turned into 0. In order to decrease the discrepancy between the inference and the training, they use regularization to make the distribution function closer to the corners of the simplex (i.e., one-hot vectors). They also enable asymmetric search, by using the original distribution values for the query vector. In contrast, our work learns both the soft and hard representation explicitly as part of an end-to-end training by using the Straight Through estimator technique [2], and exploits Euclidean distances. This results in a richer expressive power, which improves the classification and retrieval performance, as demonstrated in Sec. 4.

3. Deep Product Quantization

Architecture. The diagram of the DPQ architecture is presented in Fig. 1. The DPQ is learned on top of the embedding layer. The nature of this embedding changes according to the protocol of each benchmark, see Sec. 4. Let x be the input to the network, and let $embedding$ be the output of the embedding layer for input x (inputs are omitted for brevity). In the first step, we learn a small multilayer perceptron (MLP) on top of the embedding layer, let $s \in \mathbb{R}^{MN}$ be the output of the MLP. The vector s is then sliced to M sub-vectors, $s = [s_1, s_2, \dots, s_M]$, where each $s_m \in \mathbb{R}^N$. On top of each sub-vector, we learn a small MLP which ends in a softmax function with K outputs. We denote the probability of the k-th entry of the softmax of the MLP that corresponds to the m-th sub-vector by $p_m(k)$. Additionally, for each sub-vector, we learn a matrix, $C_m \in \mathbb{R}^{K \times D}$ (composed of K vectors in $\mathbb{R}^D$ that represent the K centroids). We denote the k-th row of the matrix C_m by $C_m(k)$. The

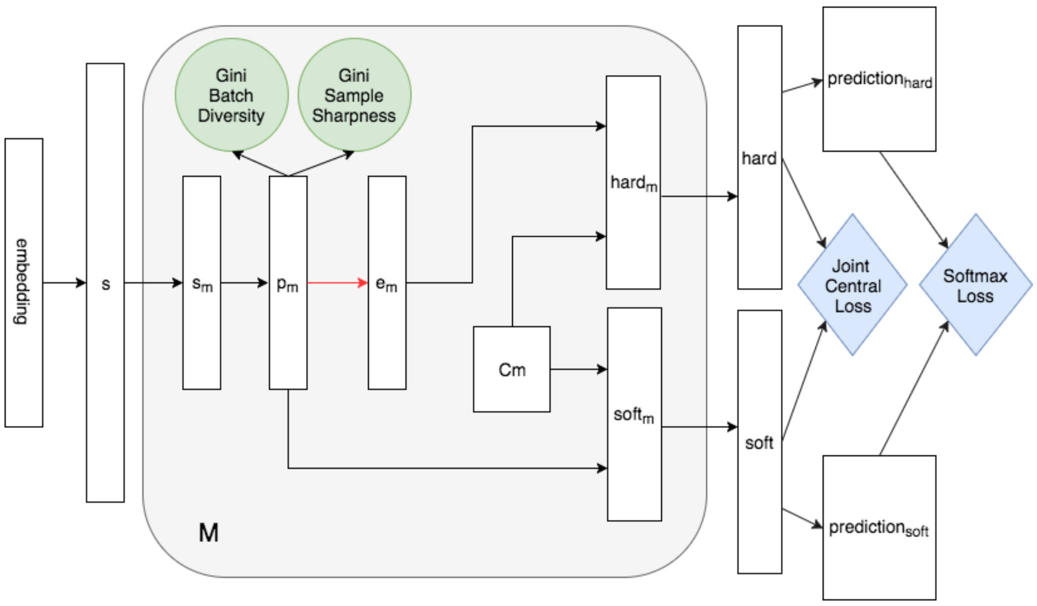

Figure 1. The architecture of the DPQ model. The Softmax Loss and the Joint Central Loss functions are denoted by the blue diamonds, and the Gini Batch Diversity and the Gini Sample Sharpness regularizations are denoted by the green circles. The red arrow is the non differential one-hot encoding transformation which requires using the Straight Through estimator in order to pass the gradients.

m-th sub-vector of the soft representation is computed as the convex combination of the rows of C_m, where the coefficients are the probability values of p_m:

$$\mathsf{soft}_m = \sum_{k=1}^{K} p_m(k) \cdot C_m(k) \tag{1}$$

Let $k^* = \text{argmax}_k\, p_m(k)$ be the index of the highest probability in p_m and let e_m be a one hot encoding, such that $e_m(k^*) = 1$ and $e_m(k) = 0$ for $k \neq k^*$. Then, the m-th sub-vector of the hard representation is computed by:

$$\mathsf{hard}_m = \sum_{k=1}^{K} e_m(k) \cdot C_m(k) = C_m(k^*) \tag{2}$$

Therefore, the m-th sub-vector of the hard representation is equal to the row in C_m that corresponds to the entry k^* with the highest probability in p_m. Since the conversion of the probability distribution p_m to a one hot encoding, e_m is not a differential operation, we employ the idea of straight-through (ST) estimator [2] to enable the back-propagation, i.e., the computation of the one hot encoding in the forward pass is performed using the $argmax$ function, however, in the backward pass, we treat the one hot encoding layer as the identity function, and pass the gradients received by the one hot encoding layer, directly to the softmax layer that computed p_m without transforming them.

The M soft sub-vectors are concatenated to the final soft representation vector, and the M hard sub-vectors are concatenated to the final hard representation vector:

$$\mathsf{soft} = [\mathsf{soft}_1, \mathsf{soft}_2, \ldots, \mathsf{soft}_M] \tag{3}$$
$$\mathsf{hard} = [\mathsf{hard}_1, \mathsf{hard}_2, \ldots, \mathsf{hard}_M] \tag{4}$$

where soft and hard are in $\mathbb{R}^{MD}$.

For classification into C classes, a fully connected layer, defined by a matrix $W \in \mathbb{R}^{MD\times C}$ and a bias vector $b \in \mathbb{R}^C$, is used to obtain prediction scores over these C classes. We denote by $prediction_{\mathsf{soft}}$ and $prediction_{\mathsf{hard}}$, the predictions given for the soft and hard representations respectively.

Loss functions. The softmax loss is applied to $prediction_{\mathsf{soft}}$ and $prediction_{\mathsf{hard}}$ and captures the requirement that the soft and hard representations classify the samples correctly. We also devise a new loss function inspired by the center loss [17], named *Joint Central Loss*.

While the softmax loss encourages the representations to be separable with respect to the classes, the center loss encourages features from the same class to be clustered together, thus improving the discriminative power of the features and contributing to the retrieval performance. The center loss learns a center vector, $o_i \in \mathbb{R}^V$, for each class i, where $V = MD$ is the size of the representation, by minimizing the distance between the representation, $r_i \in R^V$, and the vector of the corresponding class, o_{y_i}:

$$\frac{1}{2}||r_i - o_{y_i}||^2 \tag{5}$$

The motivation for the Joint Central Loss, introduced here, is to add another role to the center loss, which is decreasing the discrepancy between the soft and hard representations, thus improving the performance of the asymmetric search. This is implemented by using the same center for both the soft and hard representations, and encouraging both representations to be closer to the same centers of the classes.

Regularization DPQ uses regularization in order to ensure near uniform distribution of the samples to their corresponding clusters, for each partition M. This empowers the training to find a solution that better utilizes the clusters in the encoding. Specifically, given a batch, B, of samples, $B = (x_1, x_2, ..., x_B)$, let $p_m^i \in \mathbb{R}^K$ be the probability distribution over the clusters of the m-th sub-vector, of the i-th sample, then the following Gini Impurity related penalty is defined as:

$$\mathsf{GiniBatch}(p_m) := \sum_{k=1}^{K} \left(\frac{1}{B} \sum_{i=1}^{B} p_m^i(k) \right)^2 \tag{6}$$

This penalty achieves a maximal value of 1 if and only if there is a single cluster, k, for which $\forall i \, p_m^i(k) = 1$, and a minimal value of $\frac{1}{K}$ if and only if $\forall k : \frac{1}{B} \sum_{i=1}^{B} p_m^i(k) = \frac{1}{K}$. Therefore, by adding this penalty, the optimization is encouraged to find a solution in which the samples are distributed more evenly to the clusters.

We also add another regularization term to encourage the probability distribution of a sample i, p_m^i to be closer to a one hot encoding:

$$\mathsf{GiniSample}(p_m^i) := -\sum_{k=1}^{K} \left(p_m^i(k) \right)^2 \tag{7}$$

This term encourages the soft and hard representations of the same sample to be closer. Note that the two loss-functions may seem competing, however the first is calculated over a batch and encourages diversity within a batch, while the second is calculated per distribution of a single sample and encourages the distributions to be decisive (i.e., close to a one hot vector).

Similar forms of these regularizations have been successively shown to help improve the performance of hashing techniques in the literature [13, 7].

3.1. Inference

The DPQ method benefits from all the advantages of Product Quantization techniques. This section elaborates on how DPQ is used to create a compressed representation, fast classification, and fast retrieval in both the symmetric and asymmetric forms.

3.1.1 Compressed Representation

For a given vector, $x \in \mathbb{R}^L$, the DPQ can compress x to the hard representation. Specifically, if DPQ is using M partitions, where each partition is encoded to K clusters, let $z_m \in \{1, \ldots, K\}$ be the index of the cluster that achieves the highest probability in p_m, and let $z = (z_1, z_2, \ldots, z_M)$ be the compressed hard representation. Then, the hard representation can be perfectly reconstructed from z and C_m, and therefore it can be represented by $M \log_2(K)$ bits. The following compression ratio is achieved when using float-32 to represent x:

$$\frac{32L}{M \log_2(K)} \tag{8}$$

3.1.2 Classification

By employing Lookup Tables, one can decrease the classification time over the hard representation. Let $\mathit{prediction}_{\mathsf{hard}}[c]$ be the output of the prediction layer for class c according to the hard representation before applying the softmax operation.

$$\begin{aligned}
\mathit{prediction}_{\mathsf{hard}}[c] &= b_c + \sum_{d=1}^{MD} W_{d,c} \cdot \mathsf{hard}[d] \\
&= b_c + \sum_{m=1}^{M} \sum_{d=1}^{D} W_{(m-1)D+d,c} \cdot \mathsf{hard}_m[d] \\
&= b_c + \sum_{m=1}^{M} \sum_{d=1}^{D} W_{(m-1)D+d,c} \cdot C_m(z_m)[d]
\end{aligned}$$

By creating M Lookup Tables of $C \cdot K$ entries,

$$\mathsf{LUTC}_m[c,k] = \sum_{d=1}^{D} W_{(m-1)D+d,c} \cdot C_m(k)[d],$$

one can compute $\mathit{prediction}_{\mathsf{hard}}[c]$ efficiently by performing M additions:

$$\mathit{prediction}_{\mathsf{hard}}[c] = b_c + \sum_{m=1}^{M} \mathsf{LUTC}_m[c, z_m]$$

3.1.3 Symmetric Comparison

The fast symmetric comparison is performed by using M Lookup Tables, $\mathsf{LUTSym}_m[k_1, k_2]$ each of $\binom{K}{2}$ entries:

$$\mathsf{LUTSym}_m[k_1, k_2] = \sum_{d=1}^{D} (C_m(k_1)[d] - C_m(k_2)[d])^2$$

The distance between the hard representations of two vectors, hard^x and hard^y, with compressed hard representations z^x and z^y respectively, can be then computed by:

$$\sum_{d=1}^{MD} (\mathsf{hard}^x[d] - \mathsf{hard}^y[d])^2 = \sum_{m=1}^{M} \mathsf{LUTSym}_m[z^x_m, z^y_m]$$

3.1.4 Asymmetric Comparison

The asymmetric comparison is evaluated on the soft representation of a vector, soft^x, and on the compressed representation of a vector, z^y, that encodes the hard representation of y, hard^y. The typical use case is when a search system receives a query, computes its soft representation, but uses hard representation to encode the vectors in the database in order to reduce the memory footprint. In this scenario, it is common to compare the single soft representation of the query with many compressed hard representation of the items in the database. For this application, one can build M Lookup Tables which are specific to the vector soft^x. Each table, $\mathsf{LUTASym}_m^{\mathsf{soft}^x}$, has K entries:

$$\mathsf{LUTASym}_m^{\mathsf{soft}^x}[k] = \\ = \sum_{d=1}^{D} (C_m(k)[d] - \mathsf{soft}^x[(m-1)\cdot D + d])^2$$

Thus, allowing the comparison of soft^x and z^y by performing M additions:

$$\sum_{d=1}^{MD} (\mathsf{soft}^x[d] - \mathsf{hard}^y[d])^2 = \sum_{m=1}^{M} \mathsf{LUTASym}_m^{\mathsf{soft}^x}[z^y_m]$$

Note that the preprocessing time, preparing the LUT per query, is justified whenever the database size is much larger than K.

4. Experiments

We evaluate the performance of the DPQ method on three important tasks: single-domain image retrieval, cross-domain image retrieval, and image classification. Our method is shown to achieve state of the art results in all of them.

Single-domain category retrieval. We use the CIFAR-10 dataset to demonstrate the DPQ performance on the single-domain category retrieval task. In this task, we train a DPQ model on the training set of CIFAR-10, and use the test set to evaluate the retrieval performance by using the mean average precision (mAP) metric. To disentangle the contribution of DPQ from the base architecture of the CNN that is applied on the image, we follow the same architecture proposed by DSH [13], which was adopted by other works that were evaluated on this benchmark. The protocol of the benchmark is to measure the mAP when using $12, 24, 36$ and 48 bits to decode the database vectors. We train DPQs with $M = (2, 4, 6, 8)$ partitions and $K = 64$ centroids per part, to match our experiments with the protocol.

Method	12-bit	24-bit	36-bit	48-bit
CNNH+ [18]	0.5425	0.5604	0.5640	0.5574
DQN [3]	0.554	0.558	0.564	0.580
DLBHC [12]	0.5503	0.5803	0.5778	0.5885
DNNH [11]	0.5708	0.5875	0.5899	0.5904
DSH [13]	0.6157	0.6512	0.6607	0.675
KSH-CNN [14]	-	0.4298	-	0.4577
DSRH [20]	-	0.6108	-	0.6177
DRSCH [19]	-	0.6219	-	0.6305
BDNN [5]	-	0.6521	-	0.6653
SUBIC [7]	0.6349	0.6719	0.6823	0.6863
DPQ-Sym (ours)	**0.6831**	**0.6865**	**0.6830**	**0.6876**
DPQ-ASym (ours)	**0.6730**	**0.6919**	**0.6951**	**0.6932**

Table 1. Retrieval performance (mAP) on the CIFAR-10 dataset for a varying number of bits.

DPQ is learned on top of the embedding layer of the base network, that has $U = 500$ units. We start by splitting each embedding $F \in \mathbb{R}^U$ into M equal parts: $F = (F_1, F_2, \ldots, F_M)$ where $F_i \in \mathbb{R}^{U/M}$ (when U is not divisible by M, we discard a few units so that M would divide U). On each sub-vector, F_i, we learn a small MLP which is composed of a fully connected layer with 128 units, a ReLU activation, and a fully connected layer with 64 units. We then apply a softmax function that outputs p_m as described in Sec. 3, with $K = 64$ entries. Our cluster vectors, C_m, are chosen to be in $\mathbb{R}^{U/M}$. Therefore, both the final hard and soft representation are in $\mathbb{R}^U$.

In addition to the loss functions and regularizations described in Sec. 3, we add a weight decay to prevent the over-fitting of the base network. As shown in Tab. 1, our DPQ method achieves state of the art results in both symmetric and asymmetric retrieval. As mentioned in Sec. 3.1, both the symmetric and asymmetric methods have the same computation complexity as SUBIC [7].

One can notice that the results for the symmetric and asymmetric retrieval are very similar. An application for which the quality of the symmetric retrieval is highly important is the *all pairwise distances*. In this application, one wishes to compute all the pairwise distances of the items in the database in order to find the most similar ones. Since all of the items in the database are compressed, the asymmetric version is not available and, therefore, one must rely on the quality of the symmetric search. The expressive power of DPQ defines $\binom{K}{2}^M$ possible distances between two hard representations of vectors. In contrast, hamming distance on $M \cdot \log_2(K)$ bits, defines $M \cdot \log_2(K) + 1$ possible distances between two binary vectors. In SUBIC [7],

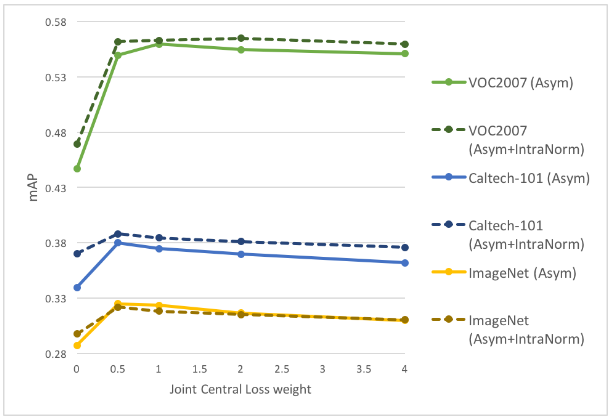

Figure 2. The retrieval performance (mAP) for the cross-domain category retrieval benchmark as a function of the Joint Central Loss weight. The DPQ model is trained on the ImageNet dataset, and is evaluated on three different datasets: VOC2007, Caltech-101, and ImageNet. As seen, The Joint Central Loss is improving the results on all the different datasets. Furthermore, the intra-normalization is improving the results for the cross-domain datasets of VOC2007 and Caltech-101, while slightly hurting the performance of ImageNet.

the hard representation is structured such that each group, $m \in \{1, \dots, M\}$, has only one bit that is on, therefore allowing for only $M+1$ possible values of distances between two hard representations of vectors. Although SUBIC [7] has not published results for the symmetric case, our hypothesis is that their symmetric results will fall behind their asymmetric results, especially for the case of $M = 2$.

Cross-domain category retrieval. In the task of cross-domain category retrieval, one evaluates a supervised hashing technique by training on a dataset with specific classes, and evaluating the retrieval results by using the mAP metric on a different dataset with a different set of classes. The authors of [16] have demonstrated the importance of using this task for the evaluation of a hashing technique, in addition to the standard single-domain category retrieval.

We follow the protocol of SUBIC [7], and train a DPQ model on vectors in $\mathbb{R}^{128}$ which were computed by applying the VGG-128 [4] model on the ILSVRC-ImageNet dataset and extracting the embedding representation. The DPQ model applies a fully connected layer with 2048 units on the input, and then applies the ReLU activation. We then split the resulting vector to 8 equal sub-vectors, each in $\mathbb{R}^{256}$. For each sub-vector, we apply the softmax function which outputs p_m, as described in Sec. 3, with $K = 256$ entries. Therefore, our DPQ encodes each vector into 64 bits in the compressed hard representation. Our cluster vectors, C_m, are chosen to be in $\mathbb{R}^{64}$.

We then evaluate the performance of hashing using DPQ for retrieval on the ImageNet test set, and on the Caltech-101 and VOC2007 datasets. We follow the protocol of SUBIC [7] and use 1000, 1000 and 2000 random query images from each dataset of Caltech-101, VOC2007, and ImageNet respectively, and use the rest as the database. Our results are presented in Tab. 2. Our method surpassed the state of the art result for the ImageNet dataset. Using the hard and soft representations directly for the VOC2007 and Caltech-101 dataset did not result in state of the art results.

Inspired by the intra-normalization technique that was presented in [1] and was used to improve the retrieval performance of a VLAD based representation, which was trained on top of SIFT features of one dataset, but then applied to another, we developed an intra-normalization technique for our soft and hard representations. Specifically, we perform $L2$ normalization for each hard_m and for each soft_m, resulting in $\mathsf{hardnorm}_m$ and $\mathsf{softnorm}_m$ respectively. We then concatenate them and produce the new hard and soft representations. Please notice that performing the $L2$ normalization to each sub-vector $m = 1 \dots M$ separately, instead of performing $L2$ normalization to the entire hard and soft representations, does not hurt our ability to use Lookup Tables to improve the search retrieval, as described in Sec. 3.1. One can simply replace the clusters of C_m with their normalized version.

The intra-normalization almost does not affect the ImageNet evaluation which is a single-domain category retrieval task. As shown in Tab. 2, the asymmetric search outperforms the symmetric search. Together with the intra-normalization technique, we achieve state of the art cross-domain performance on VOC2007.

Only for the Caltech-101 cross domain experiment, DPQ does not lead the performance chart. In this category, we match the precision of the SUBIC 2-layer method, but fall slightly behind the SUBIC 3-layer method. It is important to note that DPQ was trained only on the embedding layer of VGG-128, similarly to SUBIC 2-layer. However, SUBIC 3-layer employs the activations of the previous layer of the VGG-128 network, which, as mentioned in [16], is less specific and, therefore, expected to generalize better to other datasets. In order to study the importance of the joint central loss, we depict in Fig. 2 the mAP for the cross domain category retrieval benchmark as a function of the weight assigned to this loss. As can be seen, when training DPQ with a joint central loss of weight 0.5, a significant increase in mAP is observed across datasets. The mAP very gradually decreases as this weight further increases.

Image classification. As discussed in Sec. 3.1, one can use Lookup Tables to efficiently classify samples according to their compressed hard representation. We follow the protocol of SUBIC [7], and report the Top-1 and Top-5 accuracy on the test set of ImageNet using the 64 bit compressed

Method	VOC2007	Caltech-101	ImageNet
PQ [7]	0.4965	0.3089	0.1650
CKM[7]	0.4995	0.3179	0.1737
LSQ[7]	0.4993	0.3372	0.1882
DSH-64[7]	0.4914	0.2852	0.1665
SUBIC 2-layer [7]	0.5600	0.3923	0.2543
SUBIC 3-layer [7]	0.5588	**0.4033**	0.2810
DPQ-Sym (ours)	0.5417	0.3673	**0.3190**
DPQ-Sym + IntraNorm (ours)	0.5497	0.3731	**0.3176**
DPQ-ASym (ours)	0.5494	0.3800	**0.3250**
DPQ-ASym + IntraNorm (ours)	**0.5617**	0.3880	**0.3222**

Table 2. Retrieval performance (mAP) on the three datasets: ImageNet, Caltech, and VOC2007 where the DPQ model is trained on the ImageNet dataset only, but then evaluated on all three datasets to show cross-domain retrieval.

	ImageNet	
Method	Top-1 Accuracy	Top-5 Accuracy
VGG-128*	53.80	77.32
PQ 64-bit	39.88	67.22
CKM 64-bit	41.15	69.66
SUBIC soft*	50.07	74.11
SUBIC 64-bit	47.77	72.16
DPQ 64-bit (Ours)	**54.34**	**75.90**

Table 3. Classification performance using compressed representations. The representations marked in * are not compressed.

hard representation. As depicted in Tab. 3, our DPQ method surpasses the state of the art when classifying a compressed representation, and is the only method that achieves a Top-1 classification accuracy that is on par with the classification accuracy on top of the original features of VGG-128.

The slight improvement that is observed for the Top-1 classification implies that DPQ could have a beneficial dimensionality reduction effect. This is reminiscent of the positive effect that PCA has on the bag of word (BOW) representation, but which was not observed on the more sophisticated Fisher Vector or VLAD representations [9].

5. Discussion

In the recent literature, PQ methods have fallen behind their Hamming Distance counterparts. The main reason for this was the success of incorporating deep learning and fully supervised metric learning into the latter. By introducing a fully supervised method inspired by PQ, we are able to bring PQ techniques back to the top of the performance charts.

To our knowledge, our method is the only one to directly optimize for the retrieval of the asymmetric search, as it learns both the soft and hard representations as part of the training. Furthermore, as shown in Sec. 4, the symmetric search performance of DPQ does not fall too far behind the asymmetric search performance. This has an advantage, for example, in cases where one is interested in performing all vs. all comparisons. While the work of SUBIC [7] presented their results only for asymmetric search, one can expect that the symmetric search performance to be inferior with respect to the asymmetric search, especially for a low value of M, since the number of possible distances between two compressed representations is only $M + 1$.

Our method is also versatile in treating both retrieval and classification: it can benefit from either type of supervision (or both) and can be applied to both situations.

Especially challenging is the case where the test data differs considerably from the training data (domain shift). Paradoxically, the better the method is able to capture the structure of the training set, the more prone it is to be specific to this dataset. This is not necessarily bad, but may hinder the performance on certain benchmarks.

6. Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant ERC CoG 725974). The authors would like to thank the authors of [7] for their help with setting up the benchmarks.

References

[1] R. Arandjelovic and A. Zisserman. All about vlad. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1578–1585, 2013. 7

[2] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 3, 4

[3] Y. Cao, M. Long, J. Wang, H. Zhu, and Q. Wen. Deep quantization network for efficient image retrieval. In *AAAI*, pages 3457–3463, 2016. 3, 6

[4] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014. 7

[5] T.-T. Do, A.-D. Doan, and N.-M. Cheung. Learning to hash with binary deep neural network. In *ECCV*, pages 219–234. Springer, 2016. 6

[6] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In *CVPR*, pages 2946–2953, 2013. 2

[7] H. Jain, J. Zepeda, P. Perez, and R. Gribonval. Subic: A supervised, structured binary code for image search. In *ICCV*. 1, 3, 5, 6, 7, 8

[8] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011. 2

[9] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3304–3311. IEEE, 2010. 8

[10] Y. Kalantidis and Y. Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *CVPR*, pages 2321–2328, 2014. 2

[11] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, pages 3270–3278, 2015. 6

[12] K. Lin, H.-F. Yang, J.-H. Hsiao, and C.-S. Chen. Deep learning of binary hash codes for fast image retrieval. In *CVPR workshops*, pages 27–35, 2015. 3, 6

[13] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *CVPR*, pages 2064–2072, 2016. 3, 5, 6

[14] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081. IEEE, 2012. 6

[15] M. Norouzi and D. J. Fleet. Cartesian k-means. In *CVPR*, pages 3017–3024, 2013. 2

[16] A. Sablayrolles, M. Douze, N. Usunier, and H. Jégou. How should we evaluate supervised hashing? In *IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1732–1736, 2017. 7

[17] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In *European Conference on Computer Vision*, pages 499–515. Springer, 2016. 2, 3, 4

[18] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, volume 1, pages 2156–2162, 2014. 3, 6

[19] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *IEEE Transactions on Image Processing*, 24(12):4766–4779, 2015. 6

[20] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*, pages 1556–1564, 2015. 6