Abstract. Product quantization [1] is a popular technique for compactly encoding high dimensional vectors to facilitate scalable approximate nearest neighbor (ANN) search on massive datasets. In this paper, we present a general extension to product quantization and its recent variants [2, 3]. Rather than learning a single global product quantizer in the original ambient space, our method partitions the space and learns multiple separate product quantizers by independently minimizing the local quantization error within each partition. We investigate various partitioning schemes and show that approximate principal direction (APD) trees perform well and boost the accuracy of various existing product quantizers with a small storage and computational overhead. We evaluate our methods on ANN search benchmarks (GIST-1M, SIFT-1B, etc.) and on a codebook learning task for image classification. On the ANN search task, our new methods outperform the current state-of-the-art.

1 Introduction

Numerous applications in computer vision rely on the ability to accurately quantize high dimensional signals into a set of discrete tokens or codes, not only for compression but also for efficient search, learning statistical models and performing inference. Widely used vector quantization (VQ) techniques such as the k-means algorithm [4] form the basis of the popular bag-of-visual-words image representation [5] and methods for image retrieval [6, 7] and classification [5, 8]. There, the data vector is mapped to the nearest code vector within a pre-defined codebook. However, such 1-of-\( k \) encoding schemes do not scale to large \( k \) for high dimensional data due to excessive storage and computational costs.

Recently, product quantization (PQ) [1] was proposed as an efficient method to learn large codebooks using the notion of compositionality i.e. by treating the vector as a concatenation of sub-vectors. Considering the original space as a Cartesian product of several disjoint subspaces, traditional VQ was used to learn codebooks for sub-vectors within each subspace. Two new methods – Cartesian k-means [2] and Optimized Product Quantization [3] were subsequently proposed to improve the accuracy of PQ. Here model parameters are estimated via directly minimizing the quantization error and by searching for optimal subspace decompositions in the absence of prior knowledge about the feature space.

Binary hashing techniques [9, 10] form another line of work that also involve learning compact binary codes or strings for high dimensional vectors. Within
this category, orthogonal hashing techniques such as ITQ [11] that produces a rotated hypercube codebook can also be interpreted as a PQ method. Since each dimension is encoded with a single bit, the binary codes can be efficiently compared using Hamming distance although the accuracy is inferior to state-of-the-art PQ approaches [2, 3].

Existing product quantization methods learn a single global quantizer and optimize model parameters to minimize the global distortion or quantization error on the training data. In this paper we introduce the general idea of partitioning the ambient space into regions and then learning a bag of multiple product quantizers. Our approach involves first generating a partition and then training each quantizer separately on the subset of training data lying within each region of the partition. This allows each model to locally adapt to the data distribution within the associated region. In our setting, a vector is encoded by first assigning it to one of the quantizers. For a given bit-rate, a fixed number of code bits must encode the index of the assigned quantizer and the remaining bits are then used by the associated local quantizer, which can be based on any existing PQ method.

In the context of product quantization, we investigate various techniques for partitioning the original feature space and focus on recent algorithms with desirable theoretical properties that allow them to adapt to the low intrinsic dimension of high dimensional data. In particular, we have found that approximate principal direction (APD) trees, a recursive spatial data structure proposed recently and shown to have strong performance guarantees [12], to work well in our setting. It is also computationally efficient. We combine APD trees with four existing PQ methods and analyze their performance on standard large-scale ANN search benchmarks (SIFT-1M, GIST-1M, SIFT-1B) [1, 13] as well as on the MNIST dataset. We observe consistent improvement in retrieval accuracy for all four methods and these new variants achieve state-of-the-art results on all four datasets. Although in this paper, we do not focus on fast ANN search, we expect existing efficient indexing methods for PQ [1, 3, 13–15] to be applicable to our method as well.

1.1 Related Work

Signal quantization is a well studied topic in the image and signal processing literature [16] which is closely connected to the approximate nearest neighbor search problem [9]. Various existing methods explore quantization techniques with the aim of developing compact but accurate image representations. This includes methods for designing low-bitrate feature descriptors [17], for learning binary codes from images [10] and compressing existing feature descriptors [18] such as SIFT or applying classical dimensionality reduction [19] to them. Several related techniques have also explored the problem of designing efficient indexing schemes for fast retrieval of quantized features [14, 20] and have established the link between quantization quality and search accuracy [21].

On the other hand, space partitioning trees [22, 23, 12] are of interest to researchers designing efficient multidimensional data structures for indexing mas-
sive datasets, which will be reviewed in Section 3. Such methods are also related to existing work on tree-structured vector quantization [24, 25]. In [26], standard partitioning methods such as kd-trees [27] were used for parallelizing and speeding up codebook learning. The idea of using ensemble of local models is a widespread one in computer vision. To name a few instances, they have been used for dimensionality reduction [28], finding good non-linear embeddings [29] and in many other machine learning tasks. The novel contribution in this paper is in applying such ideas to improve the accuracy of existing product quantizers.

2 Background

In this section, we review product quantization and its extensions [1–3] and summarize recent results concerning space partitioning data structures [27, 22, 23, 12] and tree structured vector quantization [24] in Section 3.

A quantizer \( q_m : \mathbb{R}^D \rightarrow \mathbb{Z} \) maps a vector \( \mathbf{x} \in \mathbb{R}^D \) to an integer \( z \in \{0, 1, \ldots, 2^m - 1\} \) that can be encoded using \( m \)-bits (\( m \) is a positive integer). Each \( z \) is associated with a unique code \( \mathbf{c}(z) \in \mathbb{R}^D \). Given \( k \) codes \( C = \{\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{k-1}\} \in \mathbb{R}^D \), in vector quantization, 1-of-\( k \) encoding is used to assign each \( \mathbf{x} \) to its nearest code-vector, where nearness is measured by Euclidean distance. The average quantization error or distortion of \( C \) with respect to a set \( S = \{\mathbf{x}_i\}_{i=1}^N \), is equal to

\[
\mathcal{H}(q_m, S) = \mathbb{E} \left[ \min_j \|\mathbf{x} - \mathbf{c}_j\|^2 \right] = \frac{1}{N} \sum_{i}^{N} \min_{j} (\|\mathbf{x}_i - \mathbf{c}_j\|^2) \tag{1}
\]

Given a training set \( S \), the optimal codebook \( C \) is a codebook that minimizes the average distortion with respect to \( S \). However, even for modest values of \( m \), \( 2^m \) codes may be intractable to compute or store.

**Product Quantization.** Product Quantization (PQ) [1] treats \( \mathbb{R}^D \) as a Cartesian product of \( M \) subspaces each having dimensionality \( D' = \frac{D}{M} \) and trains a set of independent codebooks \( C^j \) in each of the \( D' \)-dimensional subspaces. The effective codebook can then be interpreted as \( C = C^1 \times \ldots \times C^M \). Each of the \( M \) sub-vectors are encoded into one of \( k \) codes. Thus, \( \mathbf{x} \) is encoded using \( M \log_2(k) \) bits, but can be assigned to one of \( k^M \) unique code-vectors. In PQ, individual dimensions can be grouped into subspaces according to the natural order or using structured or random permutations [1]. For 128-dimensional SIFT descriptors, typical values of \( M=16 \) and \( k=8 \) produce compact codebooks.

**Transform Coding.** An efficient quantization approach is proposed in [15] that combines transform coding (TC), data-driven allocation of bits to dimensions followed by non-uniform distortion minimizing scalar quantization. It can be thought of as product quantization in the extreme, where every subspace is one-dimensional. However, the number of scalar codes vary with dimension. After performing PCA, scalar quantization is performed in each 1D space defined
by the principal components using a Lloyd Max quantizer [16]. The greedy bit allocation strategy proposed in [15] achieves implicit dimensionality reduction by ignoring dimensions with low variance. The method is parameter free and has other benefits from the standpoint of efficient nearest neighbor search.

Optimized Product Quantization. Two new algorithms - Optimized Product Quantization (OPQ) [3] and Cartesian K-means (CKM) [2] improve upon PQ by not just optimizing the codewords or code-vectors but also by estimating an optimal rotation of the input space (Eq 2).

\[
\mathcal{H}(q_m, S) = \frac{1}{N} \sum_{i} \min_{j}(\|\hat{x}_i - \hat{c}_j\|_2^2)
\]

where \( \hat{x} = Rx \), and \( x \) is quantized to \( R^\top \hat{c} \), where \( \hat{c} \in C^1 \times C^2 \times \ldots \times C^M \), and \( R^\top R = I \) Both methods alternate between optimizing \( R \) and the code-vectors. First, \( R \) is held fixed as the code-vectors are optimized. Next, the codewords are held fixed as the optimal rotation matrix is re-estimated. As these steps are repeated, the average distortion on the training data steadily decreases. The rotation \( R \) governs the assignment of dimensions to subspaces. Optimizing \( R \) allows the training vectors to rotate, relaxing constraints on the code-vectors.

The matrix \( R \) can be initialized in multiple ways. First, initializing with the identity matrix preserves the natural ordering of feature dimensions whereas initializing with a random rotation eliminates prior knowledge. Ge et al. [3] also initialize the rotation using Eigenvalue Allocation, a method to balance variances across subspaces. This is followed by regular PQ. This method is called parametric OPQ (OPQ-P) in [3]. The code-vectors computed by OPQ-P are then used to initialize non-parametric OPQ. This is called OPQ-NP in [3]. CKM and non-parametric OPQ are exactly identical when the same initialization is used. However, for brevity, in the rest of the paper, we will refer to CKM as the method initialized with identity rotation and OPQ as the method initialized with the solution of OPQ-P.

3 Space Partitioning Data Structures

Spatial partition trees are classical data structures that have been well studied for efficient similarity search on multidimensional data [30] and search performance is known to be closely related to VQ accuracy [31]. Here, we review in brief a few recent methods that recursively construct binary trees on training data also with the goal of minimizing the average quantization distortion or error. These methods differ in the splitting rule that is used to divide the training samples at each node while learning the next level in the tree structure.

First, the classical kd-tree [27] uses axis aligned hyperplanes and the median in the chosen dimension to split the data. A random rotation or whitening transformation of the data is often used to improve kd-tree performance [32]. Random

1 uniform partitions are used as initialization during this optimization.
projection (RP) trees [22] choose a random hyperplane whereas principal component analysis (PCA) trees [23] choose a hyperplane whose normal vector is aligned to the first principal component of the data. Both PCA trees and RP trees then project the data into the chosen direction and uses the median of the projected data to split the data. PCA trees are more effective than RP trees at reducing the average diameter at the leaf nodes for a fixed tree depth \(^2\) and this is known to be related to the traditional VQ error [22].

However, PCA trees are somewhat expensive to compute since the principal component vector must be recovered at each node while learning the tree. Instead, for approximate principal direction (APD) trees [12], the principal component is approximately estimated by first choosing a random direction vector and then applying a few power-method iterations to it to obtain the orientation of the splitting hyperplane. Subsequently, the median of the projections along this direction is selected as the threshold for splitting the data. The method also has the nice property of not requiring the power-method to converge and in practice a few iterations often suffice. This makes them computationally almost as efficient as RP-trees but they have quality guarantees similar to PCA trees.

4 Proposed Method

In this section, we first describe our model and the related encoding and decoding steps. We then discuss the estimation of model parameters from training data.

Our model is a collection of local quantizers ie. a bag of product quantization models, which we denote using \(Q = \{q^i_{m_i}\}_{i=1}^L\), where each \(q^i_{m_i}\) can be any \(m_i\)-bit quantizer as defined in Section 2. We denote the parameters of \(q^i_{m_i}\) as \(\theta_i(m_i)\). As an example for CKM, we have \(L = 1\) and \(\theta_1(m) = (R, C^1, C^2, \ldots C^M)\), where \(R\) is the rotation applied to \(x\) and \(C^j\) are the subspace codebooks in the rotated space. The complete model parameters for \(Q\) can then be denoted by

\[
\Theta(Q) = [\omega, \theta_1(m_1), \theta_2(m_2), \ldots \theta_L(m_L)]
\]

(3)

where, \(\omega\) represents auxiliary information about the space partitioning model and parameters as explained later in this section and each \(\theta_i(m_i)\) parameterizes the associated \(m_i\)-bit quantizer.

**Encoding.** In our model, an input vector \(x\) is quantized by assigning it to one of the local quantizers, whose index is denoted \(i(x)\), such that \(i(x) \in \{1, 2, \ldots L\}\). With fixed bit length encoding, storing \(i(x)\) requires \(h = \lceil \log_2(L) \rceil\) bits. If the target number of bits is \(m\), then \(m_i(x) = m - h\) bits are available for use by each local product quantizer. This is constant in our case since fixed bit length is used, but could vary between different local quantizers if variable bit length encoding was used for \(i(x)\).

\(^2\) Here, diameter is defined as the distance between the furthest pair of points within any cell partition of the tree [23]
The assignment $i(x)$ can be computed using two methods. First, exhaustive search could be used by encoding $x$ using all $L$ quantizers and selecting the one that has the lowest distortion. Alternatively, a recursive binary space partitioning tree could be used to perform approximate search for the best quantizer. If each leaf node stores a pointer to one of the local quantizers, the assignment $i(x)$ could be obtained by traversing the tree and reaching a leaf node. Given $i(x)$, $x$ can then be easily quantized by the associated quantizer using $m_i(x)$ bits.

Exhaustive search is more accurate but its runtime is $O(L)$. The second approach performs the search approximately; this is $O(\log(L))$ hence faster when the tree structure is balanced. However, storing the parameters of the tree incurs an $O(L)$ storage overhead. Depending on the number of parameters at each internal node of the tree, this overhead may prevent setting $L$ to a large number in practice. All experiments in the main paper are done using exhaustive search.

Decoding. Given a $m$-bit vector $[b_h \ b_{m-h}]$, the leading $h$ bits $b_h$ are used to recover the index $i$ and code-vector corresponding to $b_{m-h}$ is retrieved from the $i$-th quantizer. The complexity of the decoding step primarily depends on the complexity of the corresponding product quantizer.

4.1 Learning

Learning the parameters of our model can be formulated as the problem of estimating the optimal number of local models $L^*$ and the parameter vector $\Theta_Q^*$ that minimizes the distortion $H(Q, S)$ on the full training set $S$. Such an optimization problem is challenging due to the discrete continuous objective function and the numerous free parameters in the model.

Instead of trying to jointly learn parameters for the complete model, we explore a simpler approach. First, we assume that $L^*$ is known. We then use an existing technique to construct a space partitioning tree of height $1 + \lceil \log_2(L^*) \rceil$ given the training dataset $S$. Theoretical analysis [12, 22, 23] shows that the tree construction algorithms are actually minimizing expected quantization error where each leaf node of the tree has a representative code-vector associated with it. Once the tree has been constructed, we use it to induce a partition of the training set into mutually exclusive subsets: $S = \{S_1 \cup S_2 \cup \ldots S_L\}$.

Next, we must select a product quantization model for each partition region and subsequently estimate its associated codebook parameters $\theta_i(m_i)$ by minimizing $H(q_{m_i}, S_i)$. This computation is performed separately on each subset. In practice, it is convenient to choose the same quantizer for all subsets. The number of quantizers $L^*$ is determined empirically in our current approach.

We now discuss one specific example of our model $Q$: APDT+TC, where APD trees are used to construct the partitions and Transform Coding is used for training the local quantizers. Unlike the original approach [15], APDT+TC performs local PCA in different regions of the data and the greedy bit allocation strategy adapts to the local variance in the data.
4.2 Subspace Decomposition and Bit Allocation

Existing PQ methods assume that the dimension $D$ is an exact multiple of the subspace dimensionality. Typically each subspace has 256 codewords represented with $b_j = 8$ bits. Our model however requires product quantizers that can output arbitrary $m - h$ bit representations where $m - h = \sum_{j=1}^{M} b_j$ but may not be a multiple of $M$. We propose a simple extension for PQ, CKM or OPQ that generates suitable values for $\{b_j\}$ for arbitrary values of $h$. We try to set $b_j$ for $j = 1 \ldots M$ to $u \in \{6, 7, 8\}$ and select the value $u^*$ that has the largest value of the remainder $m - h - u \lfloor \frac{m-h}{u} \rfloor$. Note that the last subspace may have fewer than $u^*$ bits allocated. Given $u^*$, the number of subspaces $M$ can be computed as $M = \lceil \frac{m-h}{u^*} \rceil$. We limit $u$ to the range $[6,8]$ to obtain a reasonably small number of code-vectors in each subspace.

Once $M$ and the bit allocation $\{b_j\}$ is known, we must also allocate the $D$ dimensions to these $M$ subspaces. We denote this array of allocated dimensions as $\{D'_j\}$. Initially, $d' = \lfloor \frac{D}{M} \rfloor$ dimensions are assigned to each subspace. Next, the remainder $d_r = D - Md'$ dimensions are distributed amongst the first $d_r$ subspaces, each of them receiving one extra dimension. For example, for $D = 128$ and $M = 10$, our method allocates 13 dimensions to the first eight subspaces and 12 to the remaining two. We used this greedy approach in all our experiments.

The bit allocation governs the number of codewords while the allocation of dimensions governs the dimensionality of the subspaces. Ideally, these allocations should be refined in a way that achieves the best balance ie. all the $M$ ratios $r_j = \frac{D'_j}{b_j}$ should be as equal as possible. We considered a few heuristics such as repeatedly finding the subspaces $A$ and $B$ with the minimum and maximum values of $r_j$ and reallocating one dimension from $B$ to $A$ until maximum balance is achieved. However, in practice we did not find a significant improvement from this refinement.

4.3 Analysis for the Partitioning Scheme

We tried various existing approaches for computing the partitions $\{S_1, S_2 \ldots S_L\}$ and discuss our findings in this section. Classical clustering techniques such as k-means clustering and Gaussian mixture model (GMM) learning on the whole training data are commonly applied on such tasks. However, GMM’s are fairly inefficient to train in higher dimensions\(^3\). We compare their performance with two tree structured space partitioning methods – RP trees and APD trees.

Table 1(a) lists the average distortion or quantization error of 1 million database vectors (SIFT-1M) that was obtained with four different partitioning methods, used along with Transform Coding to generate $m = 64$ bitcodes when $L = 128$. APD trees perform the best and this was also clear from the ANN search results which we report in the supplementary material. Thus, we use APD trees in all our experiments. Computationally they are much more attractive than k-means clustering which can be slow in higher dimensions or training

\(^3\) we use Gaussian components with diagonal covariances to avoid overfitting.
Table 1. (a) The average distortion of the quantized database vectors for various partitioning schemes used with Transform Coding (TC) on SIFT-1M with 64 bits. (b) APD tree partitions consistently reduce distortion for all four PQ methods. Column $T_{tr}$ indicates the training time in seconds for all eight methods.

<table>
<thead>
<tr>
<th>Type</th>
<th>Distortion</th>
<th>Type</th>
<th>Distortion</th>
<th>Type</th>
<th>Distortion</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMEANS+TC</td>
<td>1.98×10^4</td>
<td>TC</td>
<td>3.32×10^4</td>
<td>APDT+TC</td>
<td>1.95×10^4</td>
</tr>
<tr>
<td>GMM+TC</td>
<td>2.58×10^4</td>
<td>PQ</td>
<td>2.35×10^4</td>
<td>APDT+PQ</td>
<td>2.05×10^4</td>
</tr>
<tr>
<td>RPRTREE+TC</td>
<td>1.97×10^4</td>
<td>CKM</td>
<td>2.16×10^4</td>
<td>APDT+CKM</td>
<td>1.93×10^4</td>
</tr>
<tr>
<td>APDTREE+TC</td>
<td>1.95×10^4</td>
<td>OPQ</td>
<td>2.75×10^4</td>
<td>APDT+OPQ</td>
<td>1.98×10^4</td>
</tr>
</tbody>
</table>

(a) (b)

Table 1(b) lists the distortion for the SIFT-1M database for four PQ methods used along with the APD tree partitioning ($m = 64$ bits, $L = 128$). The average distortion on the quantized database vectors consistently decreases. It is worth noting that for a simple method such as TC, space partitioning lowers the quantization error considerably, from 3.32×10^4 to 1.95×10^4.

The computational cost of learning is equal to the cost of learning the tree added to that of learning the local quantizers on subsets of the training data. Table 1(b) lists the training times for all algorithms on the SIFT-1M dataset for $m = 64$ and $L = 128$. Even for large values of $L$, the APDT+ methods are faster than the baselines and this is because they are trained on subsets of the training data unlike the baselines which are trained on the whole set. In addition, these multiple quantizers can be easily trained in parallel.

5 Experimental Results

We first evaluate the proposed algorithms on Euclidean approximate nearest neighbor (ANN) search and then analyze accuracy vs storage trade-offs in our method. Finally, we report results on codebook learning for an image categorization task.

5.1 ANN Retrieval Task

In the ANN retrieval task, for each query vector, database vectors are retrieved based on their approximate Euclidean distances to the query. Two common distance metrics are – Symmetric Distance (SDC), where both the database and query vectors are quantized before computing distances, and Asymmetric Distance (ADC), where only the database vectors are quantized and compared to the original query vector [1]. Since we do not focus on speed in this paper, we used linear scan to find nearest neighbors (NN) based on the quantized distances.

Datasets. We evaluate our methods using three datasets commonly used for the ANN task [1,15,2,3]. The SIFT1M dataset contains disjoint sets of

a GMM which was at least two orders of magnitude slower on 128-dimensional data.

Table 1(b) lists the distortion for the SIFT-1M database for four PQ methods used along with the APD tree partitioning ($m = 64$ bits, $L = 128$). The average distortion on the quantized database vectors consistently decreases. It is worth noting that for a simple method such as TC, space partitioning lowers the quantization error considerably, from 3.32×10^4 to 1.95×10^4.

The computational cost of learning is equal to the cost of learning the tree added to that of learning the local quantizers on subsets of the training data. Table 1(b) lists the training times for all algorithms on the SIFT-1M dataset for $m = 64$ and $L = 128$. Even for large values of $L$, the APDT+ methods are faster than the baselines and this is because they are trained on subsets of the training data unlike the baselines which are trained on the whole set. In addition, these multiple quantizers can be easily trained in parallel.

5 Experimental Results

We first evaluate the proposed algorithms on Euclidean approximate nearest neighbor (ANN) search and then analyze accuracy vs storage trade-offs in our method. Finally, we report results on codebook learning for an image categorization task.

5.1 ANN Retrieval Task

In the ANN retrieval task, for each query vector, database vectors are retrieved based on their approximate Euclidean distances to the query. Two common distance metrics are – Symmetric Distance (SDC), where both the database and query vectors are quantized before computing distances, and Asymmetric Distance (ADC), where only the database vectors are quantized and compared to the original query vector [1]. Since we do not focus on speed in this paper, we used linear scan to find nearest neighbors (NN) based on the quantized distances.

Datasets. We evaluate our methods using three datasets commonly used for the ANN task [1,15,2,3]. The SIFT1M dataset contains disjoint sets of
Fig. 1. Results on SIFT-1M (top), GIST-1M (middle) and MNIST (bottom) using the ADC distance function. The left column shows the Euclidean 1-NN recall@N for the various methods. The right column shows mAP versus the number of bits. Here mAP is used to evaluate the accuracy of the 100-NN retrieval task.
Fig. 2. Results on SIFT-1M (top), GIST-1M (middle) and MNIST (bottom) using the SDC distance function. The left column shows the Euclidean 1-NN recall@N for the various methods. The right column shows mAP versus the number of bits. Here mAP is used to evaluate the accuracy of the 100-NN retrieval task.
128-d SIFT descriptors – 1 million base, 100k training and 10k query vectors. The GIST1M dataset also has disjoint sets of 960-d GIST features – 1 million base, 500K training and 1K query vectors whereas the SIFT1B dataset contains 1 billion base, 100 million training and 10K query vectors. As in [2], we use only the first 1 million of the training set for SIFT1B. In addition we also used the MNIST dataset of 28×28 handwritten digit images, which contains 60K training images (also treated as the base set) and 10K test images.

**Algorithms.** We compare the four baseline methods – **TC** (Transform Coding) [15], **PQ** [1] with identity rotation, **CKM** Cartesian k-means [2] with identity rotation initialization and **OPQ** [3], the non-parametric method initialized with the parametric OPQ solution. Our versions of these algorithms that use APD trees are indicated using the prefix **APDT+**.

**Parameters.** TC does not require any parameters. For PQ, CKM and OPQ, we allow each PQ subspace to have 256 centers. With a budget of $B$ bits, there are $2^B/8$ subspaces. We use 100 iterations for all methods. For OPQ, half the iterations are used for the parametric initialization ($k$-means iterations) and the remaining half for the non-parametric iterative optimization.

For all APDT+ variants, we used 20 power-method iterations while learning the tree and experiment with 4–256 partitions in all our experiments. Depending on the bit budget, a variable number of bits might be allocated to each of the PQ subspaces as explained in Section 4. For APDT+ {PQ, CKM, OPQ}, either $2^6$, $2^7$ or $2^8$ centers are selected for each PQ subspace depending on whichever gave the most balanced allocation. Since the partitioned subspaces have less data than the original training set, 20 iterations are used in the optimized product quantization steps in APDT+CKM and APDT+OPQ.

5.2 ANN Retrieval Results

In figures 1 and 2 we show the performance of the above algorithms using the asymmetric distance metric (ADC) and the symmetric distance metric (SDC) respectively. The left columns in these figures show the 1-NN Recall @ N values using 64-bit encoding. The x-axis (in log scale) is the number of nearest neighbors retrieved and the y-axis is the recall. The right columns show the mean average precision (mAP) for the various methods across different bits (from 16 to 128 bits). To compute mAP, we consider the top 100 Euclidean nearest neighbors as the true neighbors. For each database vector we compute the distance to the query and plot the precision vs. recall curve by thresholding on the distance. The area under the precision vs. recall curve gives the average precision and taking the mean across the queries gives the mean average precision (mAP).

The plots clearly show that our APDT+ variants perform much better than the existing methods across the different datasets on both Recall @ N and mAP metrics. The partitioning of the space into meaningful subspaces helps the quantization methods learn a better representation of the underlying subspace. We also see that the APDT+OPQ version consistently performs the best, given that

4 http://yann.lecun.com/exdb/mnist/
the OPQ method is the winner among the baseline methods. On the SIFT1M dataset, the Recall@ 10 ADC numbers are 60.1% and 71.6% for OPQ and APDT+OPQ respectively. For TC and APDT+TC those numbers are 53.2% and 71.3% respectively. It is interesting to note that for TC, which is a parameter free method, the performance improvement of APDT+TC is quite dramatic. The APDT+TC performance is consistent with the other APDT+ variants and outperforms the existing state-of-the-art PQ methods. This indicates that the performance improvement due to space partitioning is more significant but complementary to the improvement in optimized product quantization [3] over the original baseline [1].

The relative improvement in accuracy of APDT+ variants over the corresponding baselines is somewhat higher with SDC (Figure 2) compared to ADC although the overall accuracies with SDC are lower as expected.

Similar to SIFT-1M, the APDT+ variants have higher accuracy than the baselines on the GIST1M and MNIST datasets as well. The improvement in APDT+TC is once again quite large over TC, the corresponding baseline. The performance improvement on MNIST is particularly significant. Both the GIST1M and MNIST datasets are high dimensional. However, the distribution of GIST descriptors has large variance compared to the MNIST images which belong to ten different digits. Space partitioning is especially advantageous in this scenario, the Recall @ 1 ADC numbers for MNIST are 36.7% and 56.4% for OPQ and APDT+OPQ. Figure 4 shows the results on the SIFT1B dataset. To expedite computation, we only report the Recall @ N values. The space partitioning gives a significant boost in accuracy on this dataset as well, the Recall @ 10 numbers for CKM and APDT+CKM are 29.5% and 36.6% respectively which is an improvement of 24%.

5.3 Accuracy vs Memory overhead

In the proposed APDT+ algorithms, there is a clear trade-off in the number of header bits $h$ giving $L$ partitions vs. the number of bits $(m - h)$ used by the product quantizers. As $h$ increases, the tree depth increases resulting in finer partitions. Each partition has fewer points and the corresponding quantizers can represent them more accurately. However, there is also a memory overhead as $h$ increases (Figure 3).

In Figure 3 (a, c) we plot the Recall @ N (for N = 1 and 10) against the number of bits allocated to encode the total number of partitions. We see that as the number of partitions grow, the Recall @ N accuracy increases. However, it is important to note that as the tree depth increases the accuracy gap between the different product quantizers is smaller. This indicates that with a good partitioning of the original space, the exact choice of the product quantizer becomes less critical. The figure also shows that going from 7 to 8 bits, the Recall @ N values stop improving and start to decrease. This indicates that beyond a certain tree depth the individual quantizers tend to overfit the training data, causing the drop in accuracy. The right depth of the tree can be obtained using appropriate cross-validation on the dataset.
Fig. 3. Effect of tree depth on recall and the auxiliary memory usage of the algorithms for the corresponding values of \( h \). (a, b) shows recall@1 and (c, d) shows recall@10 for the APD-tree+ algorithms on SIFT1M using ADC with 64-bit encoding. The accuracy steadily improves with an increase in \( h \), the # header bits and so does the auxiliary memory usage. The auxiliary memory usage for APDT+CKM and APDT+OPQ is the same.

Figure 3(b, d) shows the Recall @ N vs the auxiliary memory usage costs. APDT+TC has the lowest memory cost as only the principal components associated with each subspace that has non-zero bits allocated must be stored.

If the tree structure is not used for encoding, the memory cost of APDT+PQ is the same as that of PQ. Both APDT+CKM and APDT+OPQ have the same cost and this is equal to the cost of APDT+PQ with the additional overhead of storing \( L D \times D \) rotation matrices.

5.4 Codebook Learning

In addition to ANN search, we test our approach for learning codebooks for bag of visual words (BoW) histogram features for an image categorization task. Here, we used CIFAR-10 [33], an image classification dataset with 60K labeled images (32×32 pixels) of ten categories, with 50K training and 10K test images. As in [2], we used the public implementation by Coates et. al. [8] but added our trained codebooks and feature encoders for all methods except k-means where we used the original implementation. All codebooks were trained on raw 6×6 color patches that were whitened as prescribed in [8]. Image categorization was performed using a linear SVM with \( 4k \)-dimensional BoW histogram features where two different codebook sizes were evaluated – \( k=2048 \) and 4096. The top-1 classification accuracy is reported in Table 2. All hyper-parameters including the number of partitions for APDT+ methods were determined using a coarse grid search. The best accuracy amongst the codebooks tested was obtained using CKM although the top-1 accuracy obtained with APDT+CKM and APDT+TC \((k = 4096)\) is comparable. Unlike ANN search, there isn’t a boost in accuracy.
using space partitioning although this could be due to the relatively small size of the codebooks.

6 Conclusions

We have proposed a new extension to product quantization that involves partitioning the original feature space and learning separate product quantizers on each partitions. Our experiments demonstrate that this benefits multiple variants of product quantization methods and boosts their overall accuracy for approximate nearest neighbor search and achieves significant improvements over the state-of-the-art on standard benchmarks. In this paper, we adopt a two-staged approach to learning all model parameters by first learning the tree structure and partitions and then training each local quantizer independently. Jointly optimizing the partitioning and the local quantizers is an interesting direction to investigate. In the future, we would also like to extend our method to support fast and efficient ANN search.

References


