Low-Rate Image Retrieval with Tree Histogram Coding

Vijay Chandrasekhar† David M. Chen† Zhi Li†
Gabriel Takacs† Sam S. Tsai† Radek Grzeszczuk‡ Bernd Girod†

†Information Systems Lab
Stanford University
{vijayc,dmchen,leeoz,gtakacs,sstsai,bgirod}@stanford.edu

‡ Nokia Research Center
Palo Alto, CA
radek.grzeszczuk@nokia.com

ABSTRACT

To perform image retrieval using a mobile device equipped with a camera, the mobile captures an image, transmits data wirelessly to a server, and the server replies with the associated database image information. Query data compression is crucial for low-latency retrieval over a wireless network. For fast retrieval from large databases, Scalable Vocabulary Trees (SVT) are commonly employed. In this work, we propose using distributed image matching where corresponding Tree-Structured Vector Quantizers (TSVQ) are stored on both the mobile device and the server. By quantizing feature descriptors using an optimally pruned TSVQ on the mobile device and transmitting just a tree histogram, we achieve very low bitrates without sacrificing recognition accuracy. We carry out tree pruning optimally using the BFOS algorithm and design criteria for trading off classification-error-rate and bitrate effectively. For the well known *ZuBuD* database, we achieve 96% accuracy with only $\sim$1000 bits per image. By extending accurate image recognition to such extremely low bitrates, we can open the door to new applications on mobile networked devices.

1. INTRODUCTION

Recent advances in Content Based Image Retrieval (CBIR) enable new applications on mobile devices like Mobile Augmented Reality (MAR) [1] and CD cover recognition [2]. Such applications typically use a client-server model whereby a mobile client queries a database at the server. Since it is usually infeasible to store a large image database on the client, query data must be sent to the server. Both system latency and communication cost can be reduced by sending fewer bits. Compression can help to significantly reduce the amount of data transmitted over the wireless channel and the backhaul links in a mobile network. Motivated by these considerations, we propose a novel low-bitrate method for CBIR.

1.1 Prior Work

Robust local image features are commonly used for CBIR. Popular types include Scale-Invariant Feature Transform (SIFT) [3], Gradient Location and Orientation Histogram (GLOH) [4], and Speeded-Up Robust Features (SURF) [5]. In scalable retrieval, a fast search through a large database can be achieved using a hierarchical bag-of-words classifier, such as a Scalable Vocabulary Tree (SVT) [6] or a Pyramid Match Kernel (PMK) [7]. Both the SVT and PMK train a classification tree using hierarchical k-means clustering of feature descriptors.

Mobimedia'09, September 7-9, 2009, London, UK.
Copyright 2009 ICST 978-963-9799-62-2/00/0004 ... $5.00

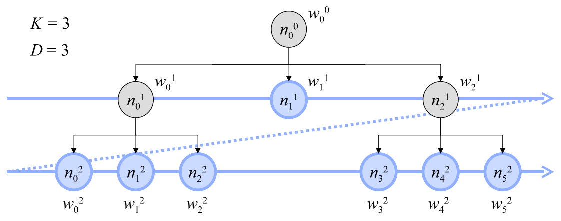


Figure 1: An example of a Tree Structured Vector Quantizer (TSVQ) with $D = 3$ and $K = 3$. A SVT is obtained by performing hierarchical k-means clustering of sample feature descriptors. The leaf nodes are marked blue in color, while the interior nodes are marked gray. The level-by-level scan order is used for run length coding in the histogram coding scheme proposed in Section 4.

Mobile CBIR entails transmitting query features from a mobile device equipped with a camera, over a wireless network, to a remote server containing the database. Several schemes have been proposed for compressing local feature descriptors in recent literature. Yeo *et al.* [8] have proposed compression of SIFT features by random projections. In our work [9], we have studied transform coding of feature descriptors. In [10], we propose a framework, CHoG (Compressed Histogram of Gradients), for creating low bitrate feature descriptors. In [11], Chen *et al.* show that storage of the Tree-Structured Vector Quantizers (TSVQ) which forms the core of SVTs is feasible on mobile devices and propose tree histogram encoding. Chen *et al.* reduce bitrate significantly by transmitting sets of features as a compressed histogram in lieu of compressing individual features. The codec exploits the fact that a bag of local image features has no inherent ordering and the tree histogram suffices for accurate classification with SVTs.

1.2 Contributions

As an extension to previous work [11], we propose methods for trading off bitrate against matching performance via optimal tree pruning BFOS [12] algorithms. Prior work [2, 6, 11] uses a large TSVQ generated by hierarchical k-means clustering of training feature descriptors. It is desirable to prune the large TSVQ to a much smaller size without adversely impacting classification accuracy of the SVT. A smaller tree has three important advantages: (1) it can

Digital Object Identifier: 10.4108/ICST.MOBIMEDIA2009.7496
http://dx.doi.org/10.4108/ICST.MOBIMEDIA2009.7496

be stored more easily at both the mobile client and server, (2) it enables significant bitrate savings in tree histogram coding because smaller tree histograms are generated and (3) it reduces the processing time on the mobile client. In this paper, we show how to systematically prune a large TSVQ using the BFOS algorithm [12] with cost functions that perform well for SVT classification. With tree histogram coding, and optimal tree pruning algorithms, the number of bits needed to query an entire image is now comparable to the number of bits in a single uncompressed SIFT feature. For the well known *ZuBuD* database, we achieve 96% accuracy with ∼1000 bits per image. For a ∼5500 *CD* cover image database, we achieve ∼90% accuracy with ∼2500 bits per image.

1.3 Overview

In Section 2, we discuss SVTs and the image retrieval pipeline. In Section 3, we discuss the distributed image matching model where identical SVTs are available on both client and server. In Section 4, we describe different schemes for compressing feature data on the client. In Section 5, we review the BFOS algorithm and propose different functionals for pruning a large SVT into smaller trees. Finally in Section 6, we present the bitrate vs. classification-error-rate trade-offs for the *ZuBuD* and *CD* datasets.

2. SCALABLE VOCABULARY TREE

A Scalable Vocabulary Tree (SVT) is obtained by performing hierarchical k-means clustering of sample feature descriptors. Two important parameters of the resulting classification tree are its branch factor, K, and tree depth, D. Initially, k-means is run on all of the training data with K cluster centers. The quantization cells are further divided by recursively repeating the subdivision process until D levels of the tree are obtained. This idea has long been familiar to the compression community as a Tree-Structured Vector Quantizer (TSVQ). The SVT aditionally comprises an inverted file at each node that references database entries (i.e., images) that contain instances of that feature descriptor.

The maximum number of nodes at the Lth level is K^L, correponding to a balanced tree. Since the tree need not be balanced, we let X_L be the actual number of nodes at level L, and the ith node in level L be denoted n_i^L. Also let the children set of node n_i^L be given by $\mathbf{C}(n_i^L)$. An example is shown in Figure 1 for $D=3$ and $K=3$. In practice, $D=6$ or $D=7$ and $K=10$ are commonly selected for good performance [6].

An image $\mathbf{I}$ is classified by quantizing its feature descriptors according to the SVT by traversing from top to bottom and greedily choosing the nearest node at each level [13]. Let the visit count $\mathbf{I}_{n_i^L}$ denote the number of descriptors of $\mathbf{I}$ which are quantized to node n_i^L. The image is compactly summarized by the tree histogram, which is the set of all visit counts. Because a parent node is visited whenever one of its children is visited, the visit count at level L can be calculated from the visit counts at level $L+1$ as

$$\mathbf{I}_{n_i^L} = \sum_{n_j \in \mathbf{C}(n_i^L)} \mathbf{I}_{n_j}. \tag{1}$$

Because of the hierarchical relationship of node counts, the entire tree histogram can be reconstructed from the counts of the leaf nodes. This property enables very efficient coding of the histogram as shown in Section 4.3.

Dissimilarity between two images is measured by comparing their tree histograms after entropy weighting. We refer to all the images with the same label as an image class. Nodes visited by fewer image classes possess greater discriminative value than nodes visited by many image classes. Nistér [6] recommends assigning to node n_i^L the weight

$$w_i^L = \begin{cases} \ln\left(|C|/|C_i^L|\right) & \text{if} \quad |C_i^L| > 0 \\ 0 & \text{if} \quad |C_i^L| = 0 \end{cases}, \tag{2}$$

where $|C|$ is the total number of image classes in the database and $|C_i^L|$ is the number of database image classes with at least one descriptor having visited node n_i^L. The weighted histogram for image $\mathbf{I}$ is a vector of all entropy-weighted visit counts through the entire SVT: $\left[w_i^L \mathbf{I}_{n_i^L}\right]$. Let the weighted histogram for a query image be given by $\mathbf{Q}$, and the mth database image by $\mathbf{D}_m$, then their dissimilarity score is

$$d\left(\mathbf{Q},\mathbf{D}_m\right) = \left\| \frac{\mathbf{Q}}{\|\mathbf{Q}\|_1} - \frac{\mathbf{D}_m}{\|\mathbf{D}_m\|_1} \right\|_1 . \tag{3}$$

By measuring $d\left(\mathbf{Q},\mathbf{D}_m\right)$ for each database image, the closest matches can be determined. We return the database image with the smallest dissimilarity score as the query result. We define classification-error-rate as the percentage of query images recognized incorrectly.

In Section 5.1, we prune a large SVT with the BFOS algorithm to create smaller unbalanced trees. Here, we extend the scoring method to unbalanced trees. To avoid favoring deeper features over shallower features in the scoring scheme, we make the path length of all features down the tree the same. For the paths that terminate early, we repeat the leaf node count down to the deepest level of the tree without splitting. However, we note that this causes only a small difference in the retrieval results.

3. CLIENT-SERVER MODEL

Many applications of CBIR require that a query be transmitted to a server for matching against a large database. One straightforward approach simply transmits the query image itself, usually with some compression (such as JPEG). In more recent work, local features descriptors are extracted at the client and these are transmitted [1, 2, 11]. Upon receipt, the server processes these feature descriptors to perform matching against a database using an SVT. In this work we propose SVT matching that is distributed between client and server. Both the client and server hold a copy of the same TSVQ, but only the server stores the inverted files. It was shown in Chen *et al.* [11] that it is feasible to store TSVQs in RAM on current mobile devices. Using our tree pruning technique, storage of the smaller pruned trees will become even more practical on mobile devices with limited RAM.

A block diagram of the proposed distributed SVT is shown in Figure 2. It allows us to compress and transmit only the information essential for the SVT matching and thereby achieves very low bitrates, while maintaining high accuracy.

4. FEATURE COMPRESSION

Since feature descriptor data must be sent over a wireless network we wish to compress these data. In this section, we discuss different schemes for transmitting features for the distributed SVT approach discussed in Section 3. For each of the schemes, features are first quantized by TSVQ.

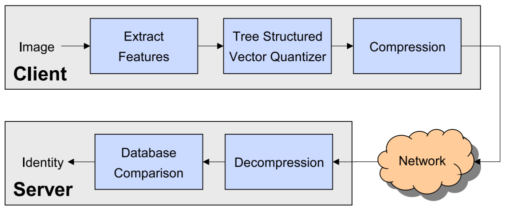


Figure 2: Distributed image matching where identical trees are used at both client and server.

4.1 Fixed-Length Coded TSVQ

We represent quantized features using a fixed-length code. If the number of leaf nodes in the SVT is N_{leaf}, each feature is represented with $\lceil \log_2(N_{\text{leaf}}) \rceil$ bits. Each feature is encoded individually and the order of features is maintained.

4.2 Entropy-Coded TSVQ

We represent quantized features using variable-length codes. Variable-length Huffman codes are trained for the alphabet of leaf nodes of the TSVQ using the statistics of the database features. Similar to Section 4.1, each feature descriptor is encoded individually.

4.3 Tree Histogram Coding

Neither of the codes in Section 4.1 and Section 4.2 exploit the fact that a "bag-of-words" approach is used and the tree histogram suffices for accurate classification. The compactness of the histogram, which enables fast search, additionally provides significant bitrate savings compared to the fixed-length and entropy coding schemes proposed in Sections 4.1 and 4.2. The gain arises since the order among the feature descriptors can be discarded.

Here, we extend the rate-efficient lossless encoder for tree histogram coding proposed in Chen *et al.* [11]. The tree histogram codec exploits the hierarchical relationship between the nodes discussed in Section 2. The histogram of leaf node counts suffices for reconstructing the entire tree histogram at the decoder. Hence, the proposed encoder transmits only the leaf histogram, while the decoder can easily reconstruct histograms at higher levels.

A typical query image produces several hundred features. In contrast, a typical SVT has a number of leaf nodes much larger than the number of features in a query image, which motivates the encoding of the length of zero-runs between positive-count nodes. SVTs need not be balanced, as illustrated in Figure 1. We traverse the leaf nodes level by level in the scan order shown in Figure 1 and run-length encode the zeros in the sequence. For example, for the tree shown in Figure 1, the sequence $[I_{n_1^1}, I_{n_0^2}, I_{n_1^2}, I_{n_2^2}, I_{n_3^2}, I_{n_4^2}, I_{n_5^2}]$ is run length encoded where $I_{n_i^L}$ is the node visit count for n_i^L. The zero-run lengths and and non-zero node counts are encoded independently by an arithmetic coder.

Since the tree histogram is a bag-of-words representation, the order among the feature descriptors is arbitrary. The number of bits saved by not transmitting the order can be significant at low bitrates. For N distinct descriptors, there are $N!$ different orderings. If each order is equally likely, it requires $\log_2(N!)$ bits to communicate a particular order. Using Stirling's approximation for large N, we find

$$\log_2(N!) \approx N(\log_2(N) - 1/\ln 2) + 1/2 \log_2(N). \quad (4)$$

Any codec that represents the node identity for each descriptor would have to spend this extra number of bits, which our codec successfully avoids. We show in Section 5 that the gains from histogram coding are significant.

4.4 Data Sets

Our experiments use two image databases. The first database is the Zurich Building Database (*ZuBuD*), consisting of 1,005 database images representing five views of $C_{ZuBuD} = 201$ different building facades in Zürich [14]. The *ZuBuD* query set consists of 115 images that are not contained in the training set. On average, each *ZuBuD* query image produces $N \approx 800$ SURF features. The second database is a CD Database (*CDD*), consisting of 5508 database images (500×500 pixels resolution) representing three different perspective views of $C_{CDD} = 1836$ different CD covers [15, 16]. In Chen *et al.* [16], we have shown how maintaining multiple perspectively distorted views of each CD image in the database improves retrieval performance. Out of this database, 100 CD covers are randomly chosen, printed on paper, inserted into jewel cases, and photographed in different environments, resulting in 100 query images [15]. On average, each *CDD* query image produces $N \approx 400$ SURF features.

4.5 Coding Results

In this section, the bitrate savings of histogram coding versus fixed-length and entropy coding are demonstrated with the *ZuBuD* dataset. We build an SVT with branch factor $K = 10$ and depth $D = 5$ with the *ZuBuD* training database. This yields a tree with 10^4 leaf nodes. Using the resulting tree and the dissimilarity metric from Section 2 we compute the error rate. The resulting classification-error-rate is $\sim$2%. Note that, once the tree is determined, the classification results and the error rate do not depend on which coder is used. We observe that fixed length coding takes $\sim$11000 bits/image, the entropy coding scheme achieves $\sim$10000 bits/image, while the histogram coding scheme achieves $\sim$3000 bits/image. These data are shown in the right-most point on the bitrate vs. classification-error-rate curves in Figure 4. Note that the performance of the histogram coding scheme is close to that predicted by Stirling's approximation in Equation 4. The histogram coding scheme provides a significant bitrate reduction.

5. TREE PRUNING

We have shown that we can save significant bitrate with tree histogram coding. We now wish to provide a mechanism for trading off classification-error-rate and bitrate so that even under the most stringent bandwidth constraints, speedy, but possibly less accurate, classification is performed. A trade-off can be provided by reducing the size of the SVT. To provide a smaller SVT that is well-suited for classification, we first build a large SVT with the standard method. This tree provides small classification-error-rate and a relatively larger bitrate. Using the BFOS [12] algorithm, we then prune the nodes which contribute the least to classification accuracy.

In addition to varying the bitrate, tree pruning also provides other benefits. Pruning prevents over-fitting and can also reduce the memory resource burden placed on the client. By creating a large tree and pruning it back so long as the classification-error-rate remains low, we can simultaneously prevent over-fitting and reduce the required memory usage. Additionally, since the optimal trees from the pruning algorithm are nested, we need to store only one tree at the client and can load the tree that suits the RAM and bitrate requirements.

We review the BFOS algorithm in Section 5.1 and discuss different functionals for pruning the SVT in Sections 5.2, 5.3 and 5.4. Finally, we discuss the bitrate vs. classification-error-rate trade-off for the different functionals in Section 6.

5.1 BFOS Algorithm

Here, we provide a brief review of the BFOS algorithm. We refer the reader to [12] for details. The BFOS algorithm was initially introduced in the context of classification and regression trees [17], but extended to variable rate TSVQ in [12]. For variable rate TSVQs, the algorithm begins with an initial tree, and prunes it back until it reaches the subtree with the fewest number of leaves that achieves a given rate distortion trade-off.

First, we define the notation that will be used in this paper. More rigorous definitions of tree, node, root, *etc.*, can be found in [12]. We define T as the full tree structure. Let t be a node of the tree, and t_0 denote the root of the tree. We define S to be a pruned subtree of T if they share the same root node, and denote this as $S \preccurlyeq T$. We denote the set of leaf nodes of a tree S as $\tilde{S}$. If a subtree consists of only the single node t, then with a slight abuse of notation, we use t to refer to the subtree t itself. We define $\psi_{\tilde{S}}$ to be a random variable that takes the node value $t \in \tilde{S}$ with probability $P(t)$. The entropy of a pruned subtree S is denoted by $H(\psi_{\tilde{S}}) = E[-\log_2 P(\psi_{\tilde{S}})] = -\Sigma_{t\in\tilde{S}} P(t)\log_2 P(t)$ The entropy of the entire tree T is denoted by $H(\psi_{\tilde{T}})$. Similarly, if $d(t)$ is the distortion at node t, then the expected distortion of a pruned subtree S is $E[d(\psi_{\tilde{S}})] = \Sigma_{t\in\tilde{S}} P(t)d(t)$, and the expected distortion of the entire tree is given by $E[d(\psi_{\tilde{T}})]$.

Real valued functions on trees and their subtrees are called tree functionals [12]. We define a tree functional on a tree S as $u(S)$. The tree functional for a node t is defined as $u(t)$. Two important properties that tree functionals may possess are linearity and monotonicity. A tree functional is linear if its value is the sum of the leaves, $u(S) = \Sigma_{t\in\tilde{S}} u(t)$. A tree functional is montonically increasing (or decreasing) if it increases (or decreases) monotonically as the tree grows. More precisely, a tree functional is monotonically non-decreasing if $u(S_1) \leq u(S_2)$ for $S_1 \preccurlyeq S_2$ and monotonically non-increasing if $u(S_1) \geq u(S_2)$ for $S_1 \preccurlyeq S_2$.

If the overall cost function of a tree (and any of its subtrees), S can be expressed as

$$u_1(S) + \mu u_2(S) \tag{5}$$

where $\mu > 0$, $u_1(S)$ is linear and monotonically non-increasing, and $u_2(S)$ is linear and monotonically non-decreasing, then the subtree of S minimizing Equation 5 can be found in linear time in the number of nodes using the BFOS algorithm [12]. The BFOS algorithm stems from the fact that, when linearity and monotonicity assumptions hold true, the optimum subtrees which minimize the Lagrangian sum of u_1 and u_2 for increasing values of μ are nested. In other words, it is possible to start with the full tree at $u(T)$ and prune back to the root at $u(t_0)$, producing a list of nested subtrees $t_0 \preccurlyeq S_n \preccurlyeq \ldots \preccurlyeq S_2 \preccurlyeq S_1 \preccurlyeq T$ which trace out the vertices of the lower boundary of the convex hull in the $u_1 - u_2$-plane.

5.2 Distortion Based Functional

In the source coding formulation of BFOS in [12], the design goal is to minimize the quantization distortion for a given rate constraint. In this case, $u_1(t)$ is chosen to be the average distortion of t, and $u_2(t)$ is chosen to be the entropy contribution of t.

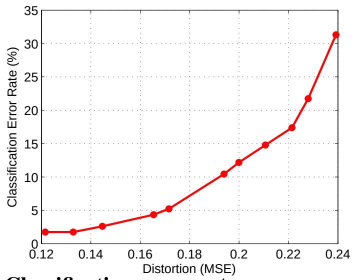


Figure 3: Classification-error-rate versus average distortion for *ZuBuD* dataset. We start with a tree of $K = 10$ and $D = 5$ and prune the tree back with BFOS algorithm with $u_1(t) = -\log_2 P(t)P(t)$ and $u_2(t) = P(t)d(t)$. We observe that classification-error-rate increases as average distortion increases.

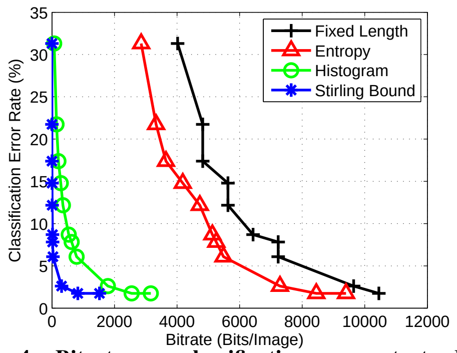


Figure 4: Bitrate vs. classification-error-rate trade-off for *ZuBuD* data set. We start with a tree of $K = 10$ and $D = 5$ and prune the tree back with BFOS algorithm with the distortion based functional. There are significant gains from histogram coding compared to entropy coding the quantized features. Note that the histogram coding rate is close to the bound provided by Stirling's approximation. The Stirling bound is obtained by subtracting the ordering rate in Equation 4 from the entropy.

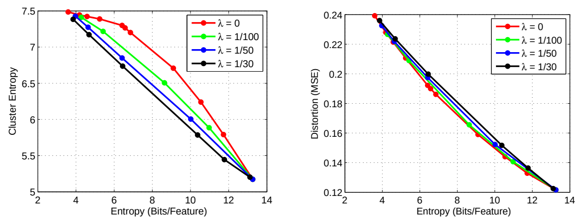
$$u_1(t) = P(t)d(t) \tag{6}$$

$$u_2(t) = -P(t)\log_2 P(t) \tag{7}$$

Note for $u_2(t)$, $H(\psi_{\tilde{S}_2}) \leq H(\psi_{\tilde{S}_1})$ for $S_2 \preccurlyeq S_1$. And for $u_1(t)$, $E[d(\psi_{\tilde{S}_2})] \geq E[d(\psi_{\tilde{S}_1})]$ for $S_2 \preccurlyeq S_1$.

Intuitively, we expect classification-error-rate to increase as distortion increases. We observe that this is indeed the case in Figure 3. The experiments are carried out on the *ZuBuD* dataset. We observe that classification-error-rate increases from 4% to 20% as the average MSE distortion increases from 0.12 to 0.21. Similar trends are also observed for the *CDD* data set.

We prune the SVT with the distortion functional and plot the trade-off of bitrate vs. classification-error-rate for the coding schemes proposed in Section 4 for the *ZuBuD* dataset. As expected, the classification-error-rate decreases as the bitrate increases in Figure 4. Entropy coding the quantized features yields upto a 33% gain in bitrate to fixed-length coding at a given classification-error-rate. We observe that there are significant gains from histogram



(*a*) (*b*)

Figure 5: Effect of varying λ on average distortion (*a*) and conditional node entropy (*b*) for the *ZuBuD* dataset. We use BFOS with the node entropy based functional. We observe that as λ increases, distortion increases at a given entropy, and the conditional node entropy decreases.

coding compared to entropy coding the quantized features. Furthermore, we observe that the histogram coding rate is close to the lower bound provided by Stirling's approximation. With histogram coding, we achieve a ∼4% classification-error-rate at bitrates as low as ∼1000 bits/image for the *ZuBuD* dataset. All bitrate classification-error trade-off curves hereon are shown for the histogram coding scheme. The bitrate classification-error trade-off for histogram coding scheme for the *CDD* dataset is shown in Figure 7 in the curves labeled $\lambda = 0$.

5.3 Node Entropy Based Functional

To improve the classification accuracy, we propose a node-entropy functional for pruning the SVT. The functional is based on the idea that nodes which are visited by feature descriptors from many different database images have less discriminatory power. Each incoming feature is quantized to one of the $t \in \tilde{T}$ leaf nodes. As stated in Section 5.1, let $\psi_{\tilde{S}}$ be a random variable that takes the node value $t \in \tilde{S}$ with probability $P(t)$. Let C be a random variable representing a class, and $|C|$ be the total number of classes. We then define the following probabilities for a pruned subtree S.

- $P(\psi_{\tilde{S}} = t) = p(t)$ for $t \in \tilde{S}$; the total probability that a feature descriptor is quantized to leaf node t.
- $P(C = c) = p(c)$ for $c = 1, ..., |C|$; the prior probability of class c.
- $P(C = c|\psi_{\tilde{S}} = t) = p(c|t)$ for $c = 1, ..., |C|$ and $t \in \tilde{S}$; the probability that a feature descriptor is quantized to leaf node t belongs to class c.
- $P(\psi_{\tilde{S}} = t|C = c) = p(t|c)$ for $c = 1, ..., |C|$ and $t \in \tilde{S}$; the probability that a feature descriptor from an image belonging to class c is quantized to leaf node t.

Given a tree and a sufficiently large training set, we can easily compute the above quantities. We now define the *node entropy* of leaf node t as

$$H(C|\psi_{\tilde{S}} = t) = H(C|t) = -\sum_{c \in C} p(c|t) \log p(c|t). \tag{8}$$

When cluster t only has features from one class, $H(C|t) = 0$. In general, $2^{H(C|t)}$ is a lower bound on the number of classes visiting leaf node t. The larger the value of $H(C|t)$, the less discriminative the feature cluster.

We define the new functionals as:

$$u_1(t) = P(t)d(t) + \lambda H(C|t)P(t) \tag{9}$$
$$u_2(t) = -P(t) \log P(t) \tag{10}$$

When $\lambda = 0$, $u_1(t)$ reverts back to the average distortion functional proposed in Section 5.2. The conditional node entropy of a pruned subtree S is obtained as the expectation of the node entropy of its leaf nodes.

$$H(C|\psi_{\tilde{S}}) = \sum_{t \in \tilde{S}} P(t)H(C|t). \tag{11}$$

The conditional entropy functional is monotonic and non-increasing i.e. $H(C|\psi_{\tilde{S}_1}) \geq H(C|\psi_{\tilde{S}_2})$ for $S_1 \preccurlyeq S_2$ [12]. Since average distortion is a monotonic non-increasing functional, the linear combination in Equation 9 is also monotonic and non- increasing.

Intuitively, the functional is chosen so that discriminative nodes, as quantified by the node entropy metric in Equation 8, are preserved in the pruning process. Between pruning two nodes that give a similar trade-off in rate-distortion, we wish to preserve the node that provides a better trade-off in node entropy.

We study the effect of varying λ and pruning the tree with the proposed functional in Equation 9. For *ZuBuD*, we sweep across the values of λ and study the trade-offs in average distortion $E[d(\psi_{\tilde{S}})]$, conditional node entropy $H(C|\psi_{\tilde{S}})$ and entropy $H(\psi_{\tilde{S}})$. We observe in Figure 5, that as λ increases the average distortion increases for a given entropy, but the conditional node entropy decreases. With values of $\lambda > 0$, a reduction in conditional node entropy $H(C|\psi_{\tilde{S}})$ at a given entropy is obtained at the expense of a higher average distortion $E[d(\psi_{\tilde{S}})]$.

Relationship to Nistér Entropy Weighting

We show the relationship of node entropy to the entropy weighting proposed by Nistér [6], as described in Section 2. Let $|C_t| = |C_L^i|$ be the number of classes that have at least one descriptor visiting node t or node n_i^L. If the prior probabilities of all classes $|C_t|$ visiting node t are equal then $H(C|t) = \log |C_t|$. The larger the value of $\log |C_t|$, the less discriminative the feature cluster. Nistér's weighting scheme then assigns a positive weighting value inversely proportional to $\log |C_t|$ given by:

$$w(t) = \log(|C|/|C_t|) \tag{12}$$

An alternative weighting scheme that takes class probabilities into account chooses

$$w(t) = \log(|C|) - H(C|t) \tag{13}$$

which in practice performs better than Nistér's weighting scheme [6] and is used for all results presented in this work. If the prior class probabilities are the same, the weighting scheme reverts back to Nistér's weighting scheme [6].

5.4 Feature Error Rate Functional

In this section, we explore a feature error rate functional for pruning the SVT. Feature error rate has been defined in prior work in the context of Classification and Regression Trees (CART) [17]. Each node in the tree is assigned to the class with the highest number of features quantized to the node. The features that do not belong to the most dominant class are considered misclassified and the feature error rate $P_e(t)$ is defined as:

$$P_e(t) = 1 - \max_{c=1...|C|} p(c|t) \tag{14}$$

We define the new functionals as:

$$u_1(t) = P(t)d(t) + \lambda P_e(t)P(t) \quad (15)$$

$$u_2(t) = -P(t)\log P(t) \quad (16)$$

When $\lambda = 0$, $u_1(t)$ reverts back to the average distortion functional proposed in Section 5.2.

Intuitively, the functional is chosen so that nodes having a smaller feature error rate as quantified by the P_e metric in Equation 15, are preserved in the pruning process. Between pruning two nodes that give a similar trade-off in rate-distortion, we wish to preserve the node that provides a better trade-off in feature error rate.

The feature error rate of a pruned subtree S is obtained as the expectation of the feature error rate of its leaf nodes.

$$P_e(\psi_{\tilde{S}}) = \sum_{t \in \tilde{S}} P(t)P_e(t). \quad (17)$$

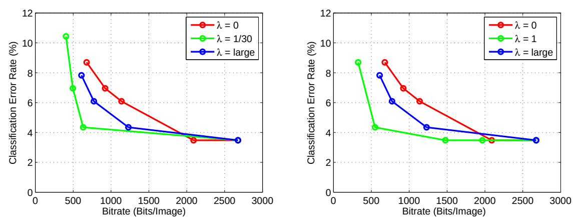
The feature error rate functional is monotonic and non-increasing i.e. $P_e(\psi_{\tilde{S}_1}) \geq P_e(\psi_{\tilde{S}_2})$ for $S_1 \preccurlyeq S_2$ [17]. Since average distortion is a monotonic non-increasing functional, the linear combination 15 is also monotonic and non- increasing.

We study the effect of varying λ and pruning the tree with the new proposed functional in Equation 15. For *ZuBuD*, we sweep across the values of λ and study the trade-offs in average distortion $E[d(\psi_{\tilde{S}})]$, feature error rate $P_e(\psi_{\tilde{S}})$ and entropy $H(\psi_{\tilde{S}})$. Similar to the node entropy functional in Section 5.3, we observe that as λ increases, the average distortion increases for a given entropy, but the feature error rate decreases. With values of $\lambda > 0$, a reduction in feature error rate $P_e(\psi_{\tilde{S}})$ at a given entropy is obtained at the expense of a higher average distortion $E[d(\psi_{\tilde{S}})]$.

6. MATCHING RESULTS

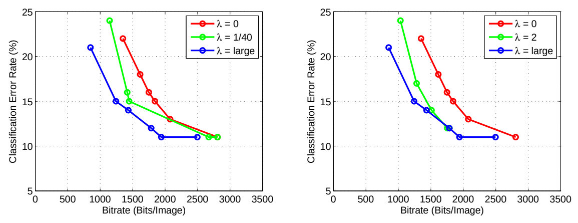
We consider both *CDD* and *ZuBuD* datasets, and plot the bitrate classification-error trade-off for both entropy coding and histogram coding schemes. For *CDD*, we start with a tree with $K = 10$ and $D = 6$ ($\sim 10^5$ leaf nodes), while for *ZuBuD* we start with a tree with $K = 10$ and $D = 5$ ($\sim 10^4$ leaf nodes), and prune the tree with the proposed tree functionals.

First, we study the bitrate classification-error trade-off for different values of λ for the *ZuBuD* data set in Figure 6. We use $\lambda = 0$ as the baseline for comparison in which case $u_1(t) = P(t)d(t)$. For the node entropy functional, we observe that as we increase $\lambda = 0$ from 0 to 1/30, the bitrate classification-error trade-off improves. For $\lambda = 1/30$, we obtain up to a 66% decrease in bitrate at a given classification-error-rate compared to $\lambda = 0$. We observe similar results for the feature error rate functional as we increase λ from 0 to 1. Alternately, we get up to a 5% decrease in classification-error-rate at a given bitrate, for both node entropy and feature error rate functionals. The improvement in performance can be attributed to the lower $H(C|\psi_{\tilde{S}})$ or $P_e(\psi_{\tilde{S}})$ at a given rate. For both node entropy and feature error functionals, we observe that the BFOS algorithm prunes off the root node early in the pruning process, for large values of λ. To obtain the points on the trade-off curve for large λ, we randomly subsample the feature set and plot the bitrate vs. classification-error-rate trade-off. For the *ZuBuD* data set, random subsampling performs better than the distortion based functional for $\lambda = 0$. However, the drawback of random subsampling is that the full SVT needs to be stored on the client. In addition to the bitrate savings, significant savings in memory can also be obtained by using the node entropy or feature error rate functionals



(*a*) (*b*)

Figure 6: Rate classification-error trade-off for *ZuBuD* data set for the histogram coding scheme for the node entropy functional (*a*) and feature error rate functional (*b*). A tree with $K = 10$ and $D = 5$ is pruned with the BFOS algorithm and node entropy and feature error rate based functionals for different values of λ. For a given classification-error-rate, upto ∼66% reduction in bitrate is obtained using node entropy and feature error rate functionals compared to the distortion functional for $\lambda = 0$.



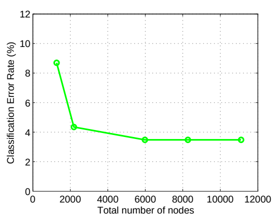
(*a*) (*b*)

Figure 7: Rate classification-error trade-off for *CDD* data set for the histogram coding schemes for the node entropy functional (*a*) and feature error rate functional (*b*). A tree with $K = 10$ and $D = 6$ is pruned with the BFOS algorithm and node entropy and feature error rate based functionals for different values of λ. For a given classification-error-rate, upto ∼25% reduction in bitrate is obtained using node entropy and feature error rate functionals compared to the distortion functional for $\lambda = 0$.

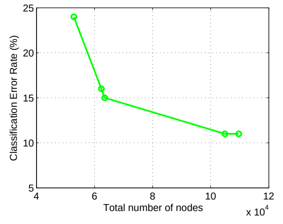
as shown in Figure 8. At one operating point, we achieve a ∼4% classification-error-rate at ∼1000 bits/image.

Similar observations are made for the *CDD* database in Figure 7. The *CDD* data set is more challenging because of the larger database size and more difficult query images. For the node entropy functional, at $\lambda = 1/40$, we achieve upto ∼25% decrease in bitrate at a given classification-error-rate, compared to $\lambda = 0$. For the feature error rate functional, at $\lambda = 2$, we achieve upto ∼25% decrease in bitrate at a given classification-error-rate, compared to $\lambda = 0$. Random subsampling corresponding to large λ performs on par with node entropy functional for $\lambda = 1/40$ and the feature error rate functional for $\lambda = 2$. However, we can save up to 2× in memory compared to storing the full tree as shown in Figure 8. At one operating point, we achieve a ∼10% classification-error-rate at ∼2500 bits/image.

Finally, we plot the memory classification-error trade-off for the node entropy functional for $\lambda = 1/30$ and $\lambda = 1/40$ for the *ZuBuD* and *CDD* datasets respectively. Similar memory classification-error trade-off curves are obtained for the feature error rate functionals. The memory required to store the SVT on client is directly proportional to the number of nodes in the tree. Compared to storing the full tree, we can obtain a 2-3× reduction in memory on the client,



(a)



(b)

Figure 8: Memory classification-error rate trade-off for the *ZuBuD* dataset (a), and the *CDD* dataset (b). We prune the tree with the BFOS algorithm with the node entropy functionals with λ = 1/30 for the *ZuBuD* dataset and λ = 1/40 for the *CDD* dataset. Note that we can get a 2-3$\times$ reduction in memory with little increase in classification-error-rate.

with little increase in the classification-error-rate. The savings in memory can be critical if the client is a mobile device with very limited RAM.

7. FUTURE WORK

In this work, we calculate the classification-error-rate by only considering the closest database candidate. Typically, top database candidates are further subjected to pairwise comparison using a ratio test and Geometric Consistency Checking (GCC) [3] with RANSAC[18]. The classification-error-rate rate can be further reduced by carrying out GCC. However, for GCC, we need to communicate the locations of all the features in the image to the server. In [19], we discuss how feature location information can be efficiently compressed using histogram map coding. In future work, we wish to study the bitrate vs. classification-error-rate trade-offs when the feature location information is also transmitted.

8. CONCLUSION

We have been able to provide high-accuracy image matching results at very low bitrates. This is possible by redefining what information shall be transmitted between the client and the server in a CBIR system. By having an identical SVT maintained at both client and server, we can eliminate bits by discarding ordering information. The resulting tree histogram coding scheme provides significant reduction in bitrate without affecting the matching accuracy. We further extend this system to operate at various bitrates by pruning the SVT using the BFOS algorithm. Performance is improved by using the proposed highly effective pruning functions based on node entropy and feature error rate. In addition to bitrate savings, the proposed pruning methods reduce the memory requirements of the client and prevents over-fitting. For the *ZuBuD* database, the resulting system can achieve 96% matching accuracy with only $\sim$1000 bits per image. This is a comparable amount of data as storing one SIFT feature conventionally. By extending accurate image recognition to such extremely low bitrates, we can open the door to new applications on mobile networked devices.

9. REFERENCES

[1] G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W.-C. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod, “Outdoors augmented reality on mobile phone using loxel-based visual feature organization,” in *Proceedings of ACM International Conference on Multimedia Information Retrieval (ACM MIR)*, Vancouver, Canada, October 2008.

[2] S. S. Tsai, D. M. Chen, J. Singh, and B. Girod, “Rate-efficient, real-time CD cover recognition on a camera-phone,” in *Proceedings of ACM Multimedia (ACM MM)*, Vancouver, British Columbia, Canada, October 2008.

[3] D. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[4] K. Mikolajczyk and C. Schmid, “Performance evaluation of local descriptors,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.

[5] H. Bay, T. Tuytelaars, and L. V. Gool, “SURF: Speeded Up Robust Features,” in *Proceedings of European Conference on Computer Vision (ECCV)*, Graz, Austria, May 2006.

[6] D. Nistér and H. Stewénius, “Scalable recognition with a vocabulary tree,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, New York, USA, June 2006.

[7] K. Grauman and T. J. Darrell, “Pyramid match hashing: sub-linear time indexing over partial correspondences,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Minneapolis, Minnesota, June 2007.

[8] C. Yeo, P. Ahammad, and K. Ramchandran, “Rate-efficient visual correspondences using random projections,” in *Proceedings of IEEE International Conference on Image Processing (ICIP)*, San Diego, California, October 2008.

[9] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, and B. Girod, “Transform coding of feature descriptors,” in *Proceedings of Visual Communications and Image Processing Conference (VCIP)*, San Jose, California, January 2009.

[10] V. Chandrasekhar, G. Takacs, D. M. Chen, S. S. Tsai, R. Grzeszczuk, and B. Girod, “CHoG: Compressed Histogram of Gradients - A low bit rate feature descriptor,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, Florida, June 2009.

[11] D. M. Chen, S. S. Tsai, V. Chandrasekhar, G. Takacs, J. Singh, and B. Girod, “Tree histogram coding for mobile image matching,” in *Proceedings of Data Compression Conference (DCC)*, Snowbird, Utah, March 2009.

[12] P. A. Chou, T. Lookabaugh, and R.M.Gray, “Optimal pruning with applications to tree-structured source coding and modeling,” *IEEE Transactions on Information Theory*, vol. 35, no. 2, pp. 299–315, Mar 1989.

[13] G. Schindler, M. Brown, and R. Szeliski, “City-scale location recognition,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Minneapolis, Minnesota, June 2007.

[14] L. V. G. H.Shao, T. Svoboda, “Zubud-Zürich buildings database for image based recognition.” ETH Zürich, Tech. Rep. 260, 2003.

[15] D. M. Chen, S. S. Tsai, J. Singh, and B. Girod, *CDD - CD Cover Database - Query Images*, April 2008. [Online]. Available: http://msw3.stanford.edu/$\sim$dchen/CDD/index.html

[16] D. M. Chen, S. S. Tsai, V. Chandrasekhar, G. Takacs, J. Singh, and B. Girod, “Robust image retrieval using multiview scalable vocabulary trees,” in *Proceedings of Visual Communications and Image Processing Conference (VCIP)*, San Jose, California, 2009.

[17] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Wadsworth International Group, Belmont, California, 1984.

[18] M. A. Fischler and R. C. Bolles, “Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[19] S. S. Tsai, D. M. Chen, G. Takacs, V. Chandrasekhar, J. P. Singh, and B. Girod, “Location coding for mobile image retreival systems,” in *Submitted to International Mobile Multimedia Communications Conference (MobiMedia)*, London, UK, September 2009.