Open access • Proceedings Article • DOI:10.1109/ICCVW.2011.6130219

Compressing Feature Sets with Digital Search Trees — Source link

Vijay Chandrasekhar, Yuriy Reznik, Gabriel Takacs, David Chen ...+3 more authors

Institutions: Stanford University, Qualcomm, Nokia

Published on: 01 Nov 2011 - International Conference on Computer Vision

Topics: Visual Word, Image retrieval, Automatic image annotation and Histogram of oriented gradients

Related papers:

- Distinctive Image Features from Scale-Invariant Keypoints
- Product Quantization for Nearest Neighbor Search
- A Novel Content-Based Image Retrieval Technique Using Tree Matching
- Wavelet Based Image Indexing and Retrieval
- Coloring image search with coupled multi-index

Share this paper:

View more about this paper here: https://typeset.io/papers/compressing-feature-sets-with-digital-search-trees-57ilnppnb7

Compressing Feature Sets with Digital Search Trees

Vijay Chandrasekhar
Stanford University

Yuriy Reznik
Qualcomm Inc.

Gabriel Takacs
Stanford University

David M. Chen
Stanford University

Sam S. Tsai
Stanford University

Radek Grzeszczuk
Nokia Inc.

Bernd Girod
Stanford University

Abstract

State-of-the-art image retrieval pipelines are based on "bag-of-words" matching. We note that the original order in which features are extracted from the image is discarded in the "bag-of-words" matching pipeline. As a result, a set of features extracted from a query image can be transmitted in any order. A set of m unique features has $m!$ orderings, and if the order of transmission can be discarded, one can reduce the query size by an additional $\log_2(m!)$ bits. We propose a coding scheme based on Digital Search Trees that reduces size of a set of features by approximately $\log_2(m!)$ bits. We perform analysis of the scheme, and show how it applies to any set of symbols in which order can be discarded. We illustrate how the scheme can be applied to a set of low bitrate Compressed Histogram of Gradients (CHoG) descriptors.

Figure 1. Typical query image with m features. Note that the features in the image can be transmitted in any order. As a result, by clever ordering of the feature set, one can reduce the query size by $\log_2(m!)$ bits.

1. Introduction

Mobile visual applications allow users to use their camera phone to initiate search queries about objects in their visual proximity. Such applications can be used, e.g., for identifying products, comparison shopping, finding information about movies, CDs, real estate, print media or artworks. First commercial deployments of such systems include Google Goggles [14], Nokia Point and Find [23], Kooaba [21], Ricoh iCandy [12, 15, 16] and Amazon Remembers [4].

Feature compression has been identified as one of the key issues for reducing latency in such mobile visual search and augmented reality applications. The size of the data sent over the network needs to be as small as possible to offer an interactive and responsive user experience. Prior work in the field has shown that extracting descriptors on the mobile device and transmitting compressed descriptors can reduce query latency significantly [8]. Several compression schemes have been proposed in recent literature. The Compressed Histogram of Gradients (CHoG) descriptor [8], Binary Robust Independent Elementary Features (BRIEF) [5], Product Quantized SIFT descriptors [17] are some examples of low bitrate descriptors.

The focus of prior work on feature compression has been primarily around compressing each individual descriptor to obtain a compact representation. Readers are referred to [8] for a detailed survey of low bitrate descriptors. In this work, we study the problem of compressing a set of features jointly. In particular, we are interested in the problem where the order in which data are transmitted does not matter, as in the case of local image features.

State-of-the-art image retrieval pipelines are commonly based on "bag-of-words" matching, i.e., query features are vector-quantized and histograms of query and database features are compared to obtain a ranked list of database images [22]. We note that the original order in which features are extracted is discarded. As a result, a set of features extracted from a query image can be transmitted in any order. A set of m unique features has $m!$ orderings, and if the order of transmission can be discarded, we should be able to reduce the query image size by an additional $\log_2(m!)$ bits.

2011 IEEE International Conference on Computer Vision Workshops
978-1-4673-0063-6/11/$26.00 ©2011 IEEE

In prior work on compressing feature sets, Chen et al. [9] notice that a tree-based representation can be used to discard the order of elements stored in it. Chen et al. propose storing a vocabulary tree [22] on the mobile device and computing the "bag-of-words" histogram locally. Run-length encoding of the non-zero bins in the histogram results in a significant reduction in query size. One drawback of this approach is that it requires the dictionary to be stored on the mobile device, which might not be feasible on RAM constrained devices. Tsai et al. [29] propose an ordering on features based on their x, y locations in the image. A histogram map is generated based on feature locations. The histogram map is then encoded efficiently to reduce the query size.

The problem of constructing codes for unordered sets was first considered in [25]. Here, we extend the approach proposed in [25]. We use Digital Search Trees (DST) for compressing visual feature descriptors. We show how DST based techniques can be used to reduce query size by approximately $\log_2(m!)$ bits. We show that the scheme works for arbitrary input sources and does not require a dictionary to be stored on the mobile device.

1.1. Outline

In Section 2, we describe the DST algorithm for compressing a binary input sequence. We analyse the performance of the scheme. We also show how the scheme applies to input sources with arbitrary source statistics. In Section 3, we illustrate how the DST coding scheme can be applied to compressing sets of variable bitrate CHoG descriptors. Conclusions are provided in Section 4.

2. Compressing a set of words

In Section 2.1, we describe the compression problem for a set of fixed-length words produced by a symmetric memoryless source. In Section 2.2, we discuss how a set of input words can be organized into a DST and discuss different schemes for representing DSTs efficiently. Next, we discuss the compression and decompression algorithms for our problem in Section 2.3. Finally, in Section 2.5, we describe how the DST coding scheme can be used for arbitrary input sources.

2.1. Problem description

Let $\{f_1, \ldots, f_m\}$ be a set of words that we need to encode. For simplicity, we first assume that these words are binary, distinct, have the same length $|f_i| = n$, and produced by a *symmetric memoryless source* (we relax these assumptions in subsequent sections). In this model, characters "0" and "1" appear with same probability $p = 1 - p = 1/2$ regardless of their positions or order. The entropy rate of such source is 1 bit/character [10], implying, that conventional sequential encoding of words $f_1, \ldots, f_m$ will cost at least mn bits. Hereafter, we will often refer to an example set of words shown Table 1 (second column). In this case: $m = 8$, $n = 5$, and total length $mn = 8 \times 5 = 40$ bits. We show how tree-based representations can be used to reduce the number of bits.

Table 1. Example set of binary words $\{f_1, \ldots, f_m\}$.

Index i	**Word** f_i	**DST Prefix** p_i	**Suffix** s_i
1	01011	0	1011
2	00111	00	111
3	10001	1	0001
4	01010	01	010
5	10010	10	010
6	00001	000	01
7	00110	001	10
8	00000	0000	0
Bits:	$8 \times 5 = 40$	18	22

2.2. Tree based representation

In order to construct a more compact representation of the set $\{f_1, \ldots, f_m\}$, we employ a data structure known as *Digital Search Tree* [11, 13, 20]. We start with a single root node, and assume that it corresponds to an empty word. We then pick our first word f_1, and depending on the value of its first character, we add a left or right branch to the root node, and insert a new node there. We also store a pointer to f_1 in that node. With second and subsequent words, we traverse the tree starting from the root node by following their characters. Once we hit a leaf (a node with no continuation in the direction of interest), we extend the DST by creating a new node and storing a pointer to the current word in it. This process is repeated m times, so that all words from our input set $\{f_1, \ldots, f_m\}$ are inserted.

The DST structure constructed over our example set is shown in Figure 2. The paths from root to other nodes in the tree correspond to portions (prefixes) of words inserted in this structure. We list such prefixes in the third column in Table 1. The fourth column in Table 1 lists the remainders (suffixes) of each word. We observe that DST construction effectively "splits" words f_i $(i = 1, \ldots, m)$ in two parts:

$$f_i = p_i \, s_i,$$

where p_i are prefixes covered by paths in the tree, and s_i are the remaining suffixes. Overall lengths of prefixes and the suffixes will be denoted by

$$\mathrm{P}_m = \sum_{i=1}^{m} |p_i|, \quad \text{and} \quad \mathrm{S}_m = \sum_{i=1}^{m} |s_i| = m\,n - P_m \quad (1)$$

correspondingly. In our example, shown in Table 1, the overall DST path length is $P_m = 18$, and the length of the remaining suffixes is $S_m = 40 - 18 = 22$.

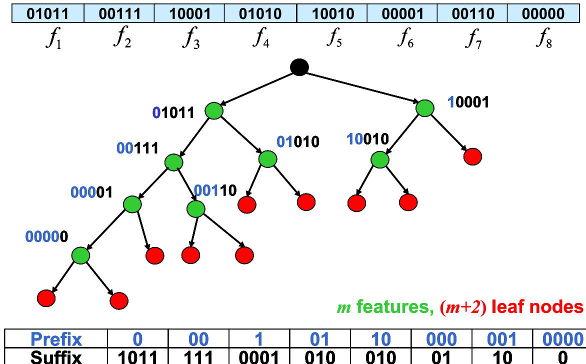


Figure 2. DST construction. Illustration of how a DST structure is constructed over our example set of words $f_1, \dots, f_8$. For m features, the tree has $m+1$ internal nodes and $m+2$ external nodes. Each word or feature corresponds to an internal node in the tree. The prefixes and suffixes for each word are shown in blue and black respectively.

In passing, we note that the order in which words $\{f_1, \dots, f_m\}$ are inserted may affect the number of characters that become "absorbed" by the tree. However, it does not change the average statistics of the tree (such as the expected path length $\mathbf{E}\, P_n$), which we will show is key to achieving the $\log(m!)$ ordering gain.

Our next task is to encode the structure of the DST efficiently. More specifically, we need to encode the shape of the binary tree. This tree contains $i = m+1$ nodes: m nodes associated with input words + one root node. We further add external or leaf nodes to the tree, as shown in Figure 2. This results in a tree with $m+1$ internal nodes or $m+2$ external nodes. We discuss two different techniques for representing the tree structure.

2.2.1 Tree traversal

One simple technique for representing the tree is to scan it recursively using pre-order tree traversal [26], and assigning labels "1" to the internal nodes, and "0" to external ones (see Figure 3). Such a sequence contains $2i+1$ digits, and serves as a unique representation of a tree with i nodes [19, 30]. We call the resulting sequence of labels an x-sequence. The x-sequence may serve as a code to represent the DST, but we show how a more compact representation can be achieved. For the example in Figure 2, a pre-order travel produces the sequence 11111000100100110000, with 1 and 0 representing internal and external nodes respectively.

In general, it is known, that the total number of possible rooted binary trees with i internal nodes is given by the Catalan number [19]:

$$C_i = \frac{1}{i+1}\binom{2i}{i}, \qquad (2)$$

implying, that a tree can be uniquely represented by only

$$\lceil \log_2 C_i \rceil \sim 2\,i - \tfrac{3}{2}\log_2 i + O(1) \quad \text{[bits]}. \qquad (3)$$

We next briefly describe one possible coding technique [30] that achieves this rate.

2.2.2 Zaks tree enumeration algorithm

The Zaks tree enumeration algorithm is used to generate an index of the tree structure. The algorithm is briefly reproduced here, and for more details, readers are referred to [30]. Given an x-sequence for a tree, we produce a list of positions of symbols "1" in it. We will call it a z-sequence $z = z_1, \dots, z_i$. For example, for a sequence $x = 1111100010010011000$, corresponding to a tree in Figure 3, we produce: $z = 1, 2, 3, 4, 5, 9, 12, 15, 16$. We next define a rule for incremental reduction of z-sequences. Let j^* be the largest j, such that $z_j = j$. By $z^* = z_1^*, \dots, z_{i-1}^*$ we will denote a new sequence that omits value z_{j^*}, and subtracts 2 from all subsequent values in the original sequence:

$$z_j^* = \begin{cases} z_j, & j = 1, \dots, j^* - 1; \\ z_{j+1} - 2, & j \geqslant j^*. \end{cases}$$

Then, a lexicographic index (or *Zaks rank*) of a tree is recursively computed as follows [30]:

$$\mathrm{index}(z) = \begin{cases} 1, & \text{if} \quad j^* = i; \\ a_{i,j^*} + \mathrm{index}(z^*), & \text{if} \quad j^* < i, \end{cases} \qquad (4)$$

where

$$a_{i,j} = \frac{j+2}{2i-j}\binom{2i-j}{i-j-1}, \quad 0 \leqslant j \leqslant i-1$$

are some constants (see Table 2).

For example, for the tree in Figure 3, Zaks ranking algorithm (4) produces:

$$\begin{aligned}
\mathrm{index}(1,2,3,4,5,9,12,15,16) &= a_{9,5} + \mathrm{index}(1,2,3,4,7,10,13,14) \\
\mathrm{index}(1,2,3,4,7,10,13,14) &= a_{8,4} + \mathrm{index}(1,2,3,5,8,11,12) \\
\mathrm{index}(1,2,3,5,8,11,12) &= a_{7,3} + \mathrm{index}(1,2,3,6,9,10) \\
\mathrm{index}(1,2,3,6,9,10) &= a_{6,3} + \mathrm{index}(1,2,4,7,8) \\
\mathrm{index}(1,2,4,7,8) &= a_{5,2} + \mathrm{index}(1,2,5,6) \\
\mathrm{index}(1,2,5,6) &= a_{4,2} + \mathrm{index}(1,3,4) \\
\mathrm{index}(1,3,4) &= a_{3,1} + \mathrm{index}(1,2) \\
\mathrm{index}(1,2) &= 1;
\end{aligned}$$

resulting in

$$\begin{aligned}
\mathrm{index}(1,2,3,4,5,9,12,15,16) &= a_{9,5} + a_{8,4} + a_{7,3} + a_{6,3} \\
&\quad + a_{5,2} + a_{4,2} + a_{3,1} + 1 \\
&= 154 + 110 + 75 + 20 \\
&\quad + 14 + 4 + 3 + 1 \\
&= 381.
\end{aligned}$$

Table 2. Coefficients $a_{i,j}$ used in lexicographic enumeration of trees.

ij	0	1	2	3	4	5	6	7	8
1	1								
2	2	1							
3	5	3	1						
4	14	9	4	1					
5	42	28	14	5	1				
6	132	90	48	20	6	1			
7	429	297	165	75	27	7	1		
8	1430	1001	572	275	110	35	8	1	
9	4862	3432	2002	1001	429	154	44	9	1

The code of this tree is a $\lceil \log_2 C_{m+1} \rceil = \lceil \log_2 C_9 \rceil = 13$ bits-long binary record of its index:

$$\mathrm{Bin}_{\lceil \log_2 C_{m+1} \rceil}(\text{index}) = \mathrm{Bin}_{13}(381) = 0000101111101.$$

As easily observed, this code is shorter than the $2i+1 = 19$ bits required for the pre-order traversal representation.

We are now ready to describe the remaining steps in our coding scheme for sets of words.

2.3. Compression and Decompression Algorithms

Given a set of m words $\{f_1, \ldots, f_m\}$, the proposed algorithm performs the following operations:

1. Build, encode, and transmit DST structure over the input set $\{f_1, \ldots, f_m\}$.
2. Scan the tree recursively, and define a canonical order of nodes and the corresponding prefixes $p_{i_1}, \ldots, p_{i_m}$ in the DST.
3. Encode and transmit suffixes according to same order $s_{i_1}, \ldots, s_{i_m}$.

The construction of the DST structure and its encoding is performed as discussed in previous sections. To define a canonical order of nodes we use the standard pre-order tree traversal [26], and assign each node a serial number, starting with 0, assigned to the root node (see Figure 3). As we reach a j-th node during the traversal, we can also recover the prefix of a word f_{i_j} that was inserted in it. This produces an order $i_1, \ldots, i_m$ in which prefixes of all words from our set can be retrieved from the tree. We omit the root node in this sequence. For example, for a tree in Figure 2, this produces the ordering $i_1 = 1, i_2 = 2, i_3 = 6, i_4 = 8, i_5 = 7, i_6 = 4, i_7 = 3, i_8 = 5$. In order to transmit information about corresponding suffixes, we simply arrange and encode them in the same order: $s_{i_1}, \ldots, s_{i_m}$. Any standard source coding technique (such as Shannon, Huffman, or arithmetic codes) can be applied for this sequence.

The decoder performs the following inverse operations:

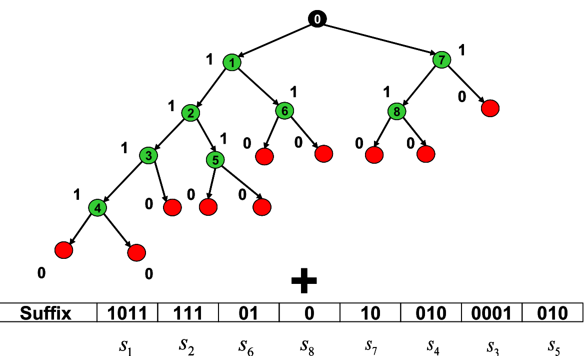


Figure 3. DST compression. Once the DST is constructed, the word set can be fully represented by (1) the structure of the tree, and (2) an ordered set of suffixes. Note that the structure of the tree captures the prefixes of all the words. For (1), the DST can be represented by a pre-order traversal. A pre-order travel produces the sequence 1111100010010011000, with 1 and 0 representing internal and external nodes respectively. The order of the traversal is indicated within each node. A more compact representation of the tree structure can be obtained by computing an index of the tree in the space of all possible trees with $m+2$ external nodes, using the Zaks algorithm. For (2), the pre-order traversal imposes an ordering on the words, and the corresponding reordered suffixes, $s_1, s_2, s_6, s_8, s_7, s_4, s_3, s_5$ are shown below the tree.

1. Decode the DST tree structure.
2. Scan nodes in the same order as encoder, and retrieve prefixes $p_{i_1}, \ldots, p_{i_m}$.
3. Sequentially decode corresponding suffixes $s_{i_1}, \ldots, s_{i_m}$, and form complete decoded words: $f_{i_j} = p_{i_j} s_{i_j}, j = 1, \ldots, m$.

We conclude our presentation of the algorithm by showing a complete code constructed for our example set of words (see Table 1, and Figures 2, 3, 4).

$$\begin{aligned} \mathrm{Code}(\{f_1, \ldots, f_m\}) &= \mathrm{Bin}_{\lceil C_{m+1} \rceil}(\text{index}), s_{i_1}, \ldots, s_{i_m} \\ &= \mathrm{Bin}_{\lceil C_9 \rceil}(381), s_1, s_2, s_6, \\ &\quad s_8, s_7, s_4, s_3, s_5 \\ &= 0000101111101, 1011, 111, 01, \\ &\quad 0, 10, 010, 0001, 010. \end{aligned}$$

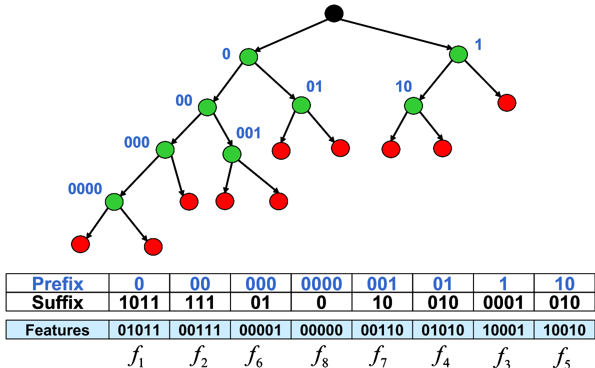


Figure 4. Decoding the DST coded data. First, the pre-order tree traversal code or Zaks index is used to reconstruct the tree. Next, the pre-order traversal of the tree is used to generate the prefixes of the features. The prefixes are combined with the corresponding suffixes to generate the final set of features (reordered).

As evident, the length of this code is $13 + 22 = 35$ bits, which is by $40 - 35 = 5$ bits shorter than the length of a straightforward sequential encoding of words in this set.

Next, we analyze the performance of the DST coding scheme.

2.4. Performance Analysis

Let us assume that input words $\{f_1, \ldots, f_m\}$ are produced by a general (not necessarily symmetric) binary memoryless source, emitting "0"s and "1"s with probabilities p and $q = 1 - p$ correspondingly. By t we will denote the total length of words in our set:

$$t = |f_1, \ldots, f_m| \,. \tag{5}$$

If we apply conventional code such as Shannon, Huffman, or arithmetic code for a *sequence* of words $f_1, \ldots, f_m$, then we know that its *average length* will satisfy [10]:

$$\bar{L}_{\text{sequence}}\,(t) = H\,t + O(1), \tag{6}$$

where

$$H = -p \log_2 p - q \log_2 q, \tag{7}$$

is the entropy of the source [10].

Consider now encoding produced by our DST-based algorithm. Here we further assume that:

$$\frac{1}{\log m} \min\{|w_1|, \ldots, |w_m|\} > \frac{1}{-\log \max\{p, q\}}.$$

This condition implies that our words $\{w_1, \ldots, w_m\}$ are longer that the *height* (longest path) of random DST [24], and so they can be uniquely parsed.

Recall, that our code consists of 2 parts: (1) encoded DST structure, occupying at most

$$L_{\text{DST}} = \lceil \log_2 C_{m+1} \rceil \leqslant \log_2 C_{m+1} + 1$$

bits, and (2) encoded sequence of suffixes. When Shannon or Huffman codes [10] are used to encode suffixes, this produces at most

$$L_{\text{suff}}(S_m) \leqslant H\,S_m + 1$$

bits, where S_m is the total length of all suffixes, and H is the entropy of the source. Consequently, the average length of a code for suffixes, will satisfy

$$\mathbf{E}L_{\text{suff}}(S_m) \leqslant H\,\bar{S}_m + 1,$$

where $\bar{S}_m = \mathbf{E}S_m$, is the expected length of suffixes in our set. In turn, $\bar{S}_m$ can be expressed as $\bar{S}_m = t - \bar{P}_m$, where $\bar{P}_m = \mathbf{E}P_m$ is the expected path length in the DST.

We next employ the result for the *expected path length* in a DST [13, 18, 27]:

$$\bar{P}_m = \frac{m}{H} \left[\log_2 m + A + \delta_1(m)\right] + O\left(\log m\right)\,, \tag{8}$$

where H is the entropy of the source, A is another known constant depending only on source parameter p, and $\delta_1(n)$ is a zero-mean oscillating function of small magnitude.

By plugging this result in the expression for code length of DST-based code, we obtain

$$\begin{aligned}
&\bar{L}_{\text{set}}\,(m,t) \\
&= L_{\text{DST}} + \mathbf{E}L_{\text{suff}}(S_m) \qquad (9)\\
&\leqslant \log_2 C_{m+1} + H\,\bar{S}_m + 2 \\
&= \log_2 C_m + H\,\left(t - \bar{P}_m\right) + 2 \\
&= 2\,m + O(\log m) + H\,t \\
&\qquad - H\left[\frac{m}{H}\left(\log_2 m + A + \delta_1(m)\right)\right] \qquad (10)\\
&= H\,t - m \log_2 m + m\left[2 - A - \delta_1(m)\right] + O(\log m)
\end{aligned}$$

and by further observing that

$$\log_2 m! = m \log_2 m - \frac{1}{\ln 2} m + O\left(\log m\right)$$

we can conclude that

$$\bar{L}_{\text{set}}\,(m,t) = H\,t - \log_2 m! + O(m)\,. \tag{11}$$

In other words, we observe that for a memoryless model, the use of DST-coding leads to savings of approximately $\log_2 m!$ bits.

2.5. Extension to Arbitrary or Unknown Sources

From analysis of DST and other random digital trees it is known that their expected path length $\bar{P}_m$ asymptotically (with large number of words m) approaches

$$\frac{\bar{P}_m}{m} \sim \frac{1}{H} \log m \tag{12}$$

where H is the entropy of source generating input words. This result is valid for a broad variety of sources, including memoryless, Markov, and ψ-mixing sources [28].

It turns out that this is also the reason why our DST-based scheme achieves close to $\log_2 m!$ saving in rate. This follows from the cancellation of factors H in Equation (10).

This means, that proposed DST-based scheme should work well under many different stochastic models of input sequences. It will automatically adapt to parameters of such sources by correspondingly changing the shape of the tree.

Finally, we show how the DST coding scheme can be applied for jointly encoding a set of Compressed Histogram of Gradients [7] descriptors.

3. Application to CHoG features

The DST coding scheme can be applied to any set of local feature descriptors. Here, we illustrate how it can be applied for jointly encoding a set of CHoG descriptors [8]. Note that we restrict our discussion to encoding of feature descriptor data. The location data associated with descriptors can be transmitted using standard variable-length encoding schemes, once feature descriptors are transmitted. First, we briefly review the structure of the CHoG descriptor and how it is computed.

3.1. CHoG Feature Structure

The pipeline for computing CHoG descriptors is shown in Figure 5. We first start with patches obtained from interest points (e.g., corners, blobs) at different scales. The patches at different scales are oriented along the dominant gradient. Next, we divide the scaled and oriented canonical patches into log-polar spatial bins. Then, we perform independent quantization of histograms in each spatial bin. The histogram in each spatial bin is quantized using Huffman Trees, Type Coding or Vector Quantization, and mapped to an index [8]. The resulting indices are then encoded using variable length codes, based on their different probabilities. The final bitstream of the feature descriptor is obtained by a concatenation of codes representative of histograms in each spatial bin.

3.2. Compression and Decompression Algorithm

The DST compression and decompression schemes are similar to those in Section 2.3. One key difference to note is that CHoG features are of variable length. We show how a simple extension to the algorithm allows us to handle variable length features. Note, also, that features need not be unique too. However, this is rarely the case as individual CHoG features are typically 25-100 bits. In case repeat features are observed, a small number of bits can be used to signal the count of non-unique features.

We use the same DST coding scheme described in Section 2.3. Let S be the number of variable-length Huffman codes, representing the S spatial components of each CHoG descriptor. Previously, the length of each feature was fixed, and so, it was used to determine the termination of suffix symbols for each feature. Now the decoding of S Huffman codes signal termination of each feature. The modified decoding algorithm is shown below:

1. Decode the DST tree structure
2. Scan nodes in the same order as encoder, and retrieve prefixes $p_{i_1}, \ldots, p_{i_m}$
3. Scan the suffix stream till S Huffman encoded parts of CHoG descriptor are decoded. The complete decoded feature is obtained by combining the prefix and suffix data. The process is repeated till all features f_{i_j} are decoded.

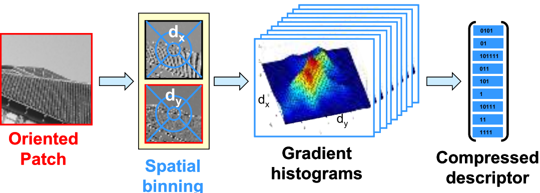


Figure 5. CHoG descriptor computation pipeline. Canonical patches around interest points are divided into log polar spatial bins. The gradient histogram in each spatial bin is represented by a variable length code.

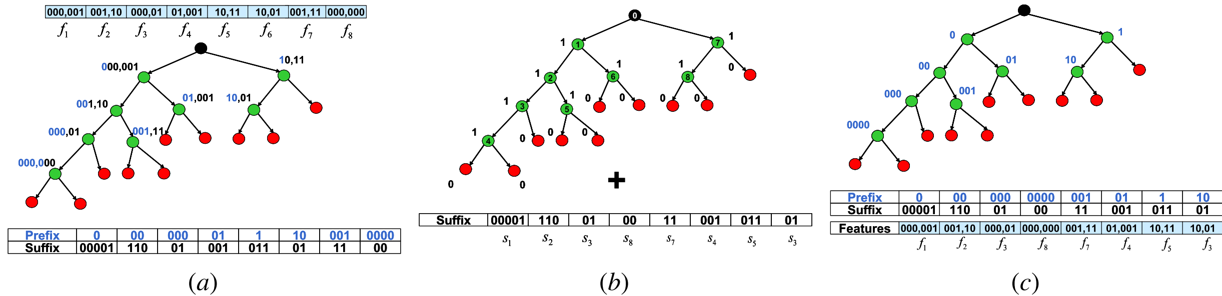
An example of CHoG DST coding is illustrated in Figure 6. In the example shown in Figure 6, each CHoG feature is obtained by concatenating $S = 2$ variable length prefix-free Huffman codes from the set ((000), (001), (01), (10), (11)).

3.3. Results

For evaluating the performance of the DST coding scheme, we use 1000 images from the *Mixed Text and Graphics* data set of the MPEG evaluation framework for "Compact Descriptors for Visual Search" [3, 2]. The images are also available as part of the Stanford Mobile Visual Search data set [6].

For feature extraction, we use a DoG interest point detector and ∼70-bit reference CHoG descriptor, which consists of 9 spatial bins, represented by 9 variable-length Huffman codes [1]. The number of features is varied by selecting features with the highest Hessian response for a given feature budget.

The results are shown in Figure 8. Here we note that the DST coding scheme reduces the data by $\sim \log_2(m!)$



(a) (b) (c)

Figure 6. Illustration of DST coding of CHoG features. Each CHoG feature is obtained by concatenating a number of variable length prefix-free Huffman codes (one for each spatial bin). In this example, each CHoG feature is obtained by concatenation of two prefix-free Huffman codes from the set $(000), (001), (01), (10), (11)$. In Figure (a), we construct the DST structure in the same fashion as before. Note that the features are now of variable length. Multiple symbols within each feature are shown as comma-delimited. Next, as shown in Figure (b), we order the features based on a pre-order traversal and transmit the tree structure, and the corresponding reordered suffixes. Finally, in the decompression step, we reconstruct the DST from the tree code. After scanning prefixes, we scan the suffix stream till we decode 2 Huffman prefix-free codes and reconstruct each feature descriptor.

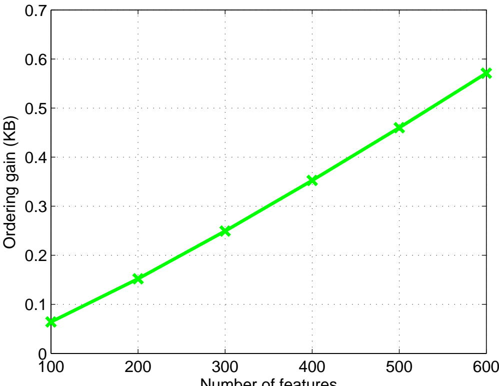


Figure 7. Reduction in data size by discarding order.

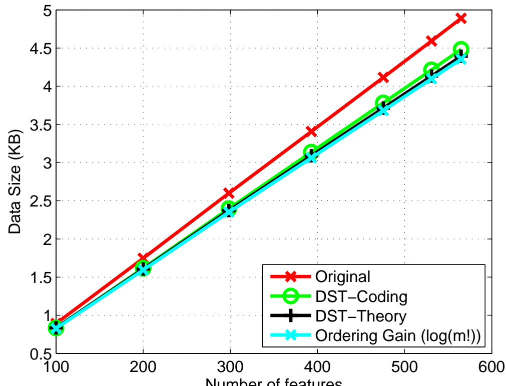


Figure 8. Results of DST coding scheme for CHoG descriptors. We note that the DST coding scheme reduces the data by $log_2(m!)$ bits, the ordering gain. Also, the performance of the DST coding scheme is close to that predicted by theory in Equation 10.

bits over a range of bitrates. At the highest query size, we can reduce the amount of data by 0.5 KB. Also, the performance of the DST coding scheme is close to that predicted by theory in Equation 10.

4. Conclusions

We propose a technique based on Digital Search Trees for encoding an unordered set of image features. We show that for a set of m unique features, we can reduce the query image size by an additional $\log_2(m!)$ bits. We perform analysis of the scheme, and show how it can be applied for encoding data from arbitrary sources. We further show how to apply the scheme for encoding a set of CHoG descriptors and show that it leads to appreciable reductions in query size.

References

[1] *Compressed Histogram of Gradients - binary release*, 2010. http://www.stanford.edu/~dmchen/mvs.html. 6

[2] Compact descriptors for visual search: Call for proposals. *ISO/IEC JTC1 SC29 WG11 output document N12201*, July 2011. 6

[3] Compact descriptors for visual search: Evaluation framework. *ISO/IEC JTC1 SC29 WG11 output document N12202*, July 2011. 6

[4] Amazon. *SnapTell*, 2007. http://www.snaptell.com. 1

[5] M. Calonder, V. Lepetit, and P. Fua. Brief: Binary robust independent elementary features. In *Proc. of European Conference on Computer Vision (ECCV)*, Crete, Greece, October 2010. 1

[6] V. Chandrasekhar, D.M.Chen, S.S.Tsai, N.M.Cheung, H.Chen, G.Takacs, Y.Reznik, R.Vedantham, R.Grzeszczuk, J.Back, and B.Girod. *Stanford Mobile Visual Search Data Set*, 2010. http://mars0.stanford.edu/mvs_images/. 6

[7] 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 *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, Florida, June 2009. 6

[8] V. Chandrasekhar, G. Takacs, D. M. Chen, S. S. Tsai, R. Grzeszczuk, Y. Reznik, and B. Girod. Compressed Histogram of Gradients: A Low Bitrate Descriptor. In *International Journal of Computer Vision, Special Issue on Mobile Vision*, 2010. Accepted. 1, 6

[9] D. M. Chen, S. S. Tsai, V. Chandrasekhar, G. Takacs, J. Singh, and B. Girod. Tree histogram coding for mobile image matching. In *Proc. of IEEE Data Compression Conference (DCC)*, Snowbird, Utah, March 2009. 2

[10] T. M. Cover and J. A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006. 2, 5

[11] J. E. G. Coffman and J. Eve. File structures using hashing functions. *Communications of the ACM*, 13(7):427–436, 1970. 2

[12] B. Erol, E. Antúnez, and J. Hull. Hotpaper: multimedia interaction with paper using mobile phones. In *Proc. of the 16th ACM Multimedia Conference*, New

York, NY, USA, 2008. 1

[13] P. Flajolet and R. Sedgewick. Digital search trees revisited. *SIAM Journal of Computing*, 15:748–767, 1986. 2, 5

[14] Google-Goggles. 2009. www.google.com/mobile/goggles/. 1

[15] J. Graham and J. J. Hull. Icandy: a tangible user interface for itunes. In *Proc. of CHI '08: Extended abstracts on human factors in computing systems*, Florence, Italy, 2008. 1

[16] J. J. Hull, B. Erol, J. Graham, Q. Ke, H. Kishi, J. Moraleda, and D. G. V. Olst. Paper-based augmented reality. In *Proc. of the 17th International Conference on Artificial Reality and Telexistence (ICAT)*, Washington, DC, USA, 2007. 1

[17] H. Jegou, M. Douze, and C. Schmid. Product Quantization for Nearest Neighbor Search. *Accepted to IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010. 1

[18] P. Kirschenhofer and H. Prodinger. Some further results on digital search trees. *Lecture Notes in Computer Science*, 229:177–185, 1986. 5

[19] D. Knuth. *The Art of Computer Programming. Fundamental Algorithms. Vol. 1*. Addison-Wesley, Reading MA, 1968. 3

[20] D. Knuth. *The Art of Computer Programming. Sorting and Searching. Vol. 3*. Addison-Wesley, Reading MA, 1973. 2

[21] Kooaba. *Kooaba*, 2007. http://www.kooaba.com. 1

[22] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, New York, USA, June 2006. 1, 2

[23] Nokia. *Nokia Point and Find*, 2006. http://www.pointandfind.nokia.com. 1

[24] B. Pittel. Asymptotic growth of a class of random trees. *Annals of Probability*, 18:414–427, 1985. 5

[25] Y. Reznik. Coding sets of words. In *Proc. of IEEE Data Compression Conference (DCC)*, Snowbird, Utah, March 2011. 2

[26] R. Sedgewick. *Algorithms. Parts 1-4. Fundamentals, Data Structures, Sorting, Searching*. Addison-Wesley, Reading MA, 1998. 3, 4

[27] W. Szpankowski. A characterization of digital search trees from the successful search viewpoint. *Theoretical Computer Science*, 85:117–134, 1991. 5

[28] W. Szpankowski. *Average case analysis of algorithms on sequences*. Wiley, 2001. 6

[29] S. S. Tsai, D. M. Chen, G. Takacs, V. Chandrasekhar, J. P. Singh, and B. Girod. Location coding for mobile image retreival systems. In *Proc. of International Mobile Multimedia Communications Conference (MobiMedia)*, London, UK, September 2009. 2

[30] S. Zaks. Lexicographic generation of ordered trees. *Theoretical Computer Science*, 10:63–82, 1980. 3