Forward Index Compression for Instance Retrieval in an Augmented Reality Application

Qi Wang* — *IBM* — NY, USA — qiwang@ibm.com

Michal Siedlaczek* — *New York University* — NY, USA — michal.siedlaczek@nyu.edu

Yen-Yu Chen† — CA, USA — yyc211@gmail.com

Michael Gormish† — *Clarifai* — CA, USA — gormish@ieee.org

Torsten Suel — *New York University* — NY, USA — torsten.suel@nyu.edu

***Abstract*—Instance retrieval systems are widely used in applications such as robot navigation, medical diagnosis, and augmented reality. Blippar is a company that creates compelling augmented reality experiences or provides you with the tools to build your own. In this paper we focus on one of the company's augmented-reality applications, with which users are able to point their phone cameras at different objects in order to receive information about the objects in real time. In this paper, we provide what we believe to be the first study of forward index compression techniques for such instance retrieval systems. First, we perform an analysis of real-world data from a large-scale commercial instance retrieval system, run by Blippar focusing on augmented reality. Then we propose an entropy-based lossless compression strategy. Experiments show that our proposed Huffman-based approach outperforms a variety of other compression techniques, while also increasing overall system efficiency slightly.**

***Index Terms*—Instance Retrieval, Index Compression, Retrieval Efficiency, Augmented Reality**

I. Introduction

Content-based instance retrieval has shown significant potential in both industry applications and research. It plays an important role in commercial image search engines, augmented reality, medical image databases, etc. There are two main approaches to content-based instance retrieval: SIFT-based and CNN-based methods [1]. The Scale Invariant Feature Transform (SIFT) [2] is used to describe important patches around key points of images, and known to be effective for identifying similar instances between the images. SIFT-based methods usually utilize a bag-of-words (BoW) model. The idea is to quantize local descriptors into much shorter "visual words", and thus represent each image as a vector of words, the same way a document is represented in text retrieval. A convolutional neural network (CNN) is a hierarchical structure that has been shown to outperform hand-crafted features such as SIFT in various vision tasks. Since the work of Krizhevsky *et al.* [3] in 2012, CNN-based methods have gained more attention. Most CNN-based and SIFT-based instance retrieval systems use a multi-phase cascading ranking architecture [4]–[6], as in text retrieval [7], [8], where the first phase obtains a large candidate pool at a lower cost, while the succeeding phases refine the instances with additional features.

In search engine architectures, a forward index stores a list of words for each document. In our application, we refer to the forward index as a data structure that stores mapping from images to their correponding features. In practice, a forward index of instance features is often stored in the main memory rather than on the disk to speed up the ranking. In our application, the size of the forward index becomes a huge issue as the number of indexed images increases. This project focuses on how to reduce the memory usage of the forward index while maintaining the same level of query throughput.

CNN-based approaches have much more advantage for specific object retrieval (e.g., buildings, pedestrians) when sufficient training data is provided, while SIFT-based approaches with a large visual-word vocabulary is still very competitive in generic object retrieval and highly efficient (see Figure 6 and Table 6 in [1]). In our application, users keep moving and pointing their phone cameras at different objects in order to receive information about those objects in real time, and create 3D models and other augmented reality actions that are triggered when these objects are recognized. This requires a system that, given a set of generic objects from a camera image, retrieves their best top-1 matches with extremely low latency. Our application prioritizes efficiency, even at the cost of a slight loss in accuracy. SIFT-based approaches with a large visual-word vocabulary are suitable in this case. While our case mainly focuses on user-interface interaction for top-1 generic object recognition, Blippar also heavily utilizes more complex CNN-models when the user decides to click on the top-1 match on the screen and obtain more information of certain recognized objects for specific categories (e.g., brands, shoes, watches, cars, etc) to enhance retrieval accuracy, which is often at a higher cost of efficiency.

In particular, our contributions are as follows:

1. We perform a thorough data analysis of forward index data from a commercial instance retrieval system, run by Blippar focusing on augmented reality. Insights from the analysis then guide us in our exploration of potential compression strategies.

2. We propose a compression strategy where we first reorder the data to transform it into a more compressible format, and then apply integer-coding techniques on the reordered data.

*This author was in an internship at Blippar when the work was performed.
†This author was with Blippar when the work was performed.

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

We consider a number of different reorderings and choices of integer compression methods, including state-of-the-art entropy coders based on Huffman coding and on Asymmetric Numeral Systems (ANS) [9], [10].

3. We provide an end-to-end evaluation of our proposed compression strategy on real-world data sets, and show that with the best choice of reordering and integer coding methods, our approach achieves good compression while also slightly improving retrieval efficiency.

The remainder of the paper is organized as follows: Section 2 presents some background and discusses related work. In Section 3 we present our data analysis on the given data set. Section 4 outlines the experimental setup. Next, Section 5 discusses the overall approach, and presents the experimental evaluation of the proposed framework. Finally, Section 6 provides some concluding remarks.

II. Background

In this section, we provide some background on instance retrieval, inverted indexes, forward indexes, reordering techniques, compression techniques, and other related work.

A. Instance Retrieval

Content-based instance retrieval are mainly divided into SIFT-based and CNN-based instance retrieval methods. SIFT-based methods train a codebook of visual words offline. Based on the size of the codebook used during encoding, they can be classified into three categories: using small codebooks, using medium-sized codebooks, and using large codebooks [1]. For methods using small codebooks, the visual words are fewer than several thousands and usually exhibit inferior accuracy. Medium-sized codebooks have sizes of 10-200k. The visual words exhibit medium discriminative ability, and the inverted index and binary signatures of visual words are constructed [11]. Large codebooks have sizes of 1 million or more. The visual words demonstrate high discriminative ability, and more memory-friendly signatures are used [12]. A frequent concern with the BoW model is the lack of geometric constraints among local features. Geometric verification are often utilized as an effective filter for matching. The most well-known method for global spatial verification is RANSAC [13]. RANSAC is effective in re-ranking a subset of top-ranked instances but has efficiency problems. To efficiently and accurately incorporate spatial features in the SIFT-based framework is very important.

The CNN-based methods have three categories: hybrid method, using pre-trained CNN models and using fine-tuned CNN models [1]. For the hybrid method, a number of image patches are generated from an input image, which are fed into the network multiple times for feature extraction. Encoding and indexing are similar to SIFT-based methods. The last two categories compute the global feature with a single network pass. For the second category, the CNN models are pre-trained on some large-scale datasets like ImageNet [3] and can be expected to be directly used on target dataset. For the third category: The CNN models are fine-tuned on a training dataset [14] which has a similar distribution to the target dataset before usage.

While CNN-based methods excel in most research benchmarking datasets (Holidays [15], Ukbench [16], etc), these datasets are relatively small. It remains unknown if training CNNs on more generic and very large-scale instance-level datasets will bring further improvement [1]. Despite the usual advantages of CNN-based methods, SIFT feature is still favored in some scenarios: gray-scale instance retrieval, intense instance color changing, small object retrieval, severe occlusions of queried instance, book/CD cover retrieval, instance of rich textures, etc [1].

B. Inverted Indexes and Forward Indexes

For image retrieval, the BoW model is efficiently implemented using an inverted index [17], which is a simple data structure that allows us to find images containing particular visual words. Given a collection of N images, we assume each image is identified by a unique image ID between 0 and $N-1$. An inverted index contains many lists, where each list L_w contains a sequence of postings describing all the images where visual word w occurs in the collection. More precisely, each posting contains the ID of an image that contains the visual word w, plus an impact score for the visual word in the image. Postings in each list are usually sorted by image ID or impact score, to allow for effective compression and fast list intersection.

In the reranking phase of image retrieval, spatial information about the visual words in a candidate image, in particular the coordinates, angle (orientation), and size of the corresponding keypoints, need to be accessed to perform geometric verification. In contrast to the inverted lists, each forward list contains all the data for a certain image, that is, all the visual words in the image as well as their associated coordinates, angles, and sizes. A forward index contains all the forward lists of the images.

C. Reordering Data for Better Compression

The idea of reordering a data set for better compression has been used by researchers in several application areas. In particular, reordering of documents has been used in information retrieval systems to improve inverted index compression, access speed [18]–[20] and query processing speed [21], [22]. Compressed graph representations can also benefit significantly from reordering of the vertices [23]–[26]. In database systems, reordering of tuples is used to improve compression of database tables [27], [28]. One common and simple way to reorder data in these applications is to sort by one of the attributes or, in the case of search engine indexes and web graphs, to sort documents or nodes by their URLs. However, other more involved approaches, such as clustering or TSP-type traversal of the data, have also been proposed, and these can sometimes significantly outperform simple sorting.

In particular, data reordering is considered an important component in most column-oriented database architectures [29], [30]. In [28], Lemire and Kaser show that picking the

right column to sort is critical as the compression ratio could improve significantly (e.g., by a factor of 2 or 3). They also try to use modular and reflected Gray-code orders and Hilbert orders to outperform lexicographical order, but find them ineffective. In summary, the idea of reordering data for compression is not new, but has not been previously applied in our application domain.

D. Compression Techniques

Many different compression techniques for inverted indexes have been proposed in the literature [31]. Most techniques assume that each list of postings is first pre-processed by taking the differences between the document IDs of any two consecutive postings, assuming the list is sorted by document ID. Thus, the problem is to compress sequences of integer values that tend to be small on average, but that may follow various distributions depending on the properties of the data set. There are many different techniques for this, including Golomb Coding [17], [31], variable-byte coding [32], Simple9 [33], OptPFD [20], etc. However, these techniques have not been previously evaluated for compression of spatial information in the scenario of image retrieval. We now outline a few of these techniques.

Golomb coding encodes an integer n in two parts: a quotient q stored as a unary code, and a remainder r in binary form. For a list of integers, we first choose a parameter P, say $P = 0.69 \times avg$, where avg is the average of the integers. Then for each number n we calculate $q = \lfloor n/P \rfloor$ and $r = n \ mod \ P$.

Variable-byte coding encodes an integer n with a sequence of bytes. In each byte, the lower 7 bits are used to store a part of the binary representation of n, and the highest bit indicates if the next byte is still part of the current number. Variable-byte is easy to implement and fast to decode; however, it does not compress as well as many bit-wise methods such as Golomb coding.

Simple9 coding combines the ideas of word alignment and bit alignment. It tries to pack as many integers as possible into one 32-bit word. One word is divided into 4 status bits and 28 data bits. The data bits can be divided in 9 different ways, e.g., into two 14-bit numbers, four 7-bit numbers, or five 5-bit numbers (leaving three bits unused), etc. The status bits then store which way the data bits are divided.

PForDelta coding splits a list of integers into chunks of fixed size. For each chunk a value b is selected so that most of the integers in that chunk are less than 2^b. Those integers can be coded together using b bits each. The remaining integers are called exceptions, and are coded separately. For best performance, chunks usually have sizes that are a multiple of 32, allowing for highly optimized code for extracting 32 b-bit values at a time during decompression (where each bank of 32 values is word-aligned). As an optimization, the selection of the b can be varied for each block to ensure the smallest compressed size. This is called OptPFD proposed by Yan *et al.* [20].

Any compression method is associated with an explicit or implicit probability model for the data that has to be compressed. For example, Golomb coding and several other methods work best if the integers follow a geometric distribution. When it is hard to make a good prediction of the next value based on a single model, general entropy coding methods such as Huffman coding, Arithmetic coding, or Asymmetric Numeral Systems (ANS) coding should perform better. ANS coding is a new approach to entropy coding developed by Duda [9] that tries to combine the advantages of Huffman and Arithmetic coding. Like Arithmetic coding, ANS coding has the ability to closely match the ideal codeword lengths when amortized over a sequence of high-probability symbols, breaking the "1 bit per symbol" limit for Huffman coding. On the other hand, ANS coding is much faster than arithmetic coding, and comparable to Huffman coding in terms of decompression speed.

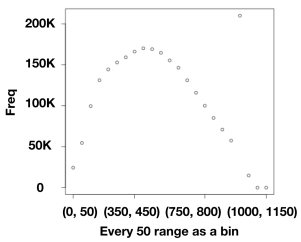

Fig. 1: Distribution of the number of key points.

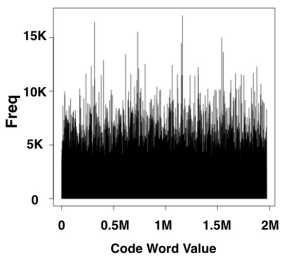

Fig. 2: Codeword frequency distribution.

III. Data Analysis

In this section, we describe our data set and its properties. We then discuss the features stored in the forward index and their distributions.

A. Data Sets

Our data is provided by Blippar, which specializes in large-scale instance retrieval and augmented reality. We build an inverted index from a collection of 2.5 million images, which is a subset of the company's production data.

B. Features

Every image in the forward index contains a certain number of key points. For each key point, we store its codeword (visual word), angle, size, and location (X and Y coordinates). Table I summarizes the statistical properties of the features across the entire index.

We now briefly describe the properties of individual features.

- **Key point count:** The average value of key points in an image is 518, while the maximum is 1130. As shown in Figure 1, values around 500 are the most frequent, except for the observed spike in the size range between 850 and 900. This is due to a soft cutoff for the maximum number of key points in each image to prioritize retrieval efficiency. While a few high quality images have more key points, most of the images have less than a thousand.

TABLE I: Statistical summary of the features in the forward index.

Features	min	1st Qu.	median	mean	3rd Qu.	max	# of bytes used	entropy	# samples
key point count	10	308	498	518	710	1,130	2	9.54	2,522,833
codeword	0	499,462	999,125	999,529	1,499,592	1,999,999	4	20.07	1,308,890,465
angle	0	16,411	32,996	33,011	49,182	65,535	2	15.69	1,308,890,465
size	230	276	354	529	542	14,712	2	9.36	1,308,890,465
X	2	77	132	136	187	317	2	8.14	1,308,890,465
Y	2	80	142	147	211	317	2	8.24	1,308,890,465

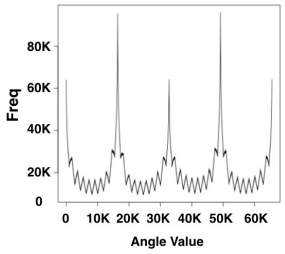

Fig. 3: Angle frequency distribution.

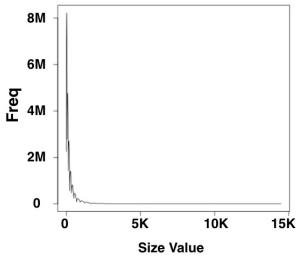

Fig. 4: Size frequency distribution.

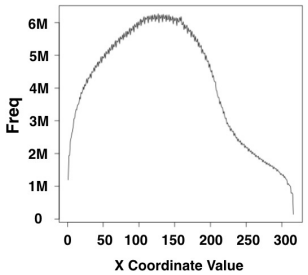

Fig. 5: The X-coordinate distribution.

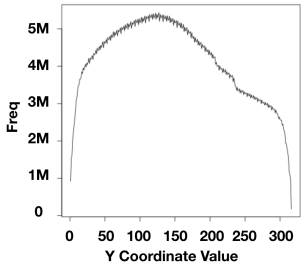

Fig. 6: The Y-coordinate distribution.

- **Codeword:** As shown in Table I, codewords range from 0 to 1999999, as we are using a vocabulary size of 2 million. Figure 2 shows the codeword frequency distribution. Although we observe no clear pattern, some codewords are clearly more frequent than others. This indicates that we might expect better compression if we remap the codeword values according to their frequency; however, the entropy of 20.07 indicates that any improvements due to this will be rather minor.
- **Angle:** Floating point numbers representing degrees (between 0 and 360) are quantized to integers between 0 and 65535 in order to avoid floating point arithmetic during reranking. As illustrated by Figure 3, the distribution of angles has 4 clear spikes, which correspond to two vertical and two horizontal directions. This is understandable, since many key points, as well as the images containing the key points, are aligned with the four axes. However, entropy is fairly large, so while the spikes are clearly visible, they may not help that much in coding. (Note that the fourth spike consists of the right edge and the left edge of the chart, as the angle wraps around.)
- **Size:** The distribution of sizes is very skewed (Figure 4). They range between 230 and 14712, while the median value is only 354. Therefore, the sizes are probably much easier to compress than codewords and angles. (of course, given there are no numbers less than 230, we can deduct 230 from each value before compressing.)
- **Coordinate:** The distributions of the X and Y coordinates are similar: they both range from 2 to 317 and are slightly skewed towards smaller values.

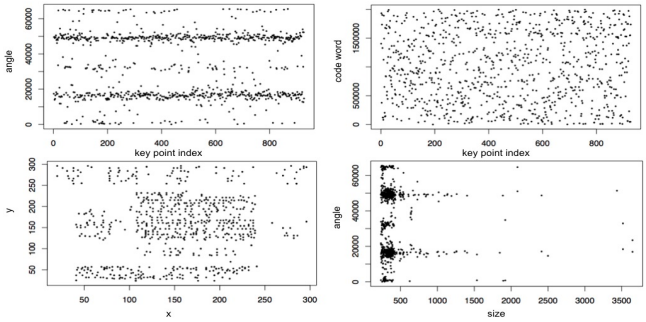

Fig. 7: Feature Distribution for Sample Image 1.

C. Distribution Within Images

Next, we show how features distribute and correlate inside individual images. Figures 7 and 8 illustrate the angle and codeword distributions, as well as the X-Y and size-angle correlations, for two random images. For angle and codeword distribution, the x-axis shows the default order in which the data items were provided at the start of this work.

The angle distribution remains consistent with the distribution across the whole index, with the four clusters along the four cardinal directions being particularly visible in the first image in Figure 7. We also observe no clear pattern in the codeword distribution: the value seems to spread evenly across the entire data range. We found the following correlation between the X and Y coordinates: when an X-coordinate occurs, it often repeats with a number of different values of Y, and vice versa. No useful pattern between size and angle is observed, apart from the fact that sizes are skewed towards smaller values.

IV. Experimental Setup

In this section, we describe the instance retrieval engine and the evaluation query set.

Instance Retrieval engine: Our instance retrieval engine is part of an Augmented Reality (AR) system currently running

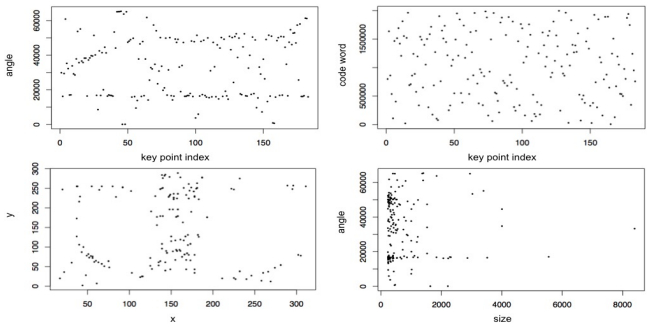

Fig. 8: Feature Distribution for Sample Image 2.

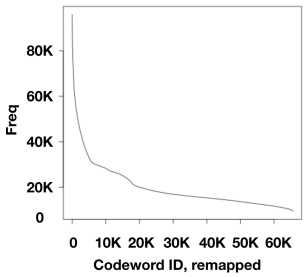

Fig. 9: Angle frequency distribution.

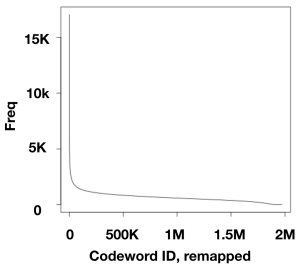

Fig. 10: Codeword frequency distribution.

features	min	1st Qu.	median	mean	3rd Qu.	max
codeword	0	499462	999125	999529	1499592	1999999
codeword*	0	198791	542558	641551	1015701	1973200
angle	0	16411	32996	33011	49182	65535
angle*	0	7897	21088	24944	40343	65535

TABLE II: Statistical Summary for orginal and remapped (*) codewords and angles.

for Blippar. The OpenCV implementation of the DoG algorithm is used for key point detection, and SIFT for descriptor generation. A 2-million codeword vocabulary was trained with an approximate k-nearest neighbor algorithm similar to [13] on a separate data set. In the candidate generation phase, we build inverted index and use a standard term-at-a-time (TAAT) implementation of top-k ranking [34] to generate 20 to 30 candidate images (instances) depending on the query. In the re-ranking phase, the codewords of the key points in the query image are matched with key points from the candidate images, using the features from the forward index.

Evaluation Query Set: A random sample of 3000 real queries was collected, submitted by users of the company's mobile application over several days.

V. Overall Approach

In this section, we describe our approach for forward index compression. We start with a set of transformations that we considered, which change the distribution of the data set to be more compressible. We then apply different encodings to the transformed index.

A. Remapping

As shown in Section 3, the frequencies of some of the codewords and angles are much larger than others. We can thus reassign codeword IDs and angles based on their frequencies, so that the more frequent values are represented by smaller numbers. After remapping, the entropy will not change, but with appropriate coding techniques, we may achieve a better compression ratio in some scenarios. Figure 9 and 10 show the remapped codeword and angle value distributions. Table II illustrates how the remapping changed the statistics of the data. The rows with * in the table indicate the results after remapping. All the angle statistics decrease, except for min and max. In case of codewords, the max also decreases slightly because the vocabulary is trained on another data set and not all codewords from the vocabulary appear in the index.

B. Reordering

Sets of integers are often presorted in ascending order so that we can then encode the differences between adjacent numbers instead of the raw numbers. In our case, the ordering of the key points (and their associated data) in the forward index is irrelevant for the second phase spatial ranking; thus, we can choose an order of key points that is best for compression. While the key points have multiple fields (features), we can only sort on one field, on which we can then use difference encoding. Based on the observations from Section 3, angle and codeword might seem to be good candidates because they are hard to compress otherwise, but this may not be obvious at this point.

C. Duplicate Reduction

As shown in Table I, the median number of key points in an image is 498, while the median value for size, X, and Y coordinates are 354, 132, and 142, respectively. This means that there are duplicate values of these three features in the majority of the images. Moreover, we observe that the same X value often occurs repeatedly with different Y values. In fact, we even observe many instances where different entries have the same X, Y, and size values (but different codewords or angles). Thus, if we sort the key points by X, then by Y, then by size, then such duplicates will be next to each other, and we can use a bit array to flag and eliminate them before coding. We can get additional benefits by coding values by their difference from the values in the previous subsection. Of course, if we decide to sort by X, Y, and size, then we cannot sort by codeword or angle as proposed in Subsection 5.B.

D. Applying Huffman and ANS Coding

Since the forward index data of each image needs to be accessed as fast as possible for second phase ranking, the complete compressed forward index is kept in main memory. We apply compression on a per-image basis so that the compressed data of any image can be efficiently fetched and decompressed for reranking. As observed before, the distribution of the features does not seem to follow a clear and simple model, and it also differs between images. Therefore, efficient entropy-based coders such as Huffman or ANS coding are attractive

choices as they can adapt to the characteristics of the data in each image. However, if we apply Huffman coding directly to the features, a separate Huffman code table has to be stored for each feature of each image, which is too much space overhead. On the other hand, a global table will not be able to adjust to different images, and will also be very large for features such as codewords and angles, slowing down decompression.

To address this issue, when encoding integers from $[0 \dots N-1]$ where smaller integers are more likely, we divide $[0 \dots N-1]$ into increasing ranges, where range sizes are powers of two. For example, the first few ranges may contain only a single integer, then two, then four, eight, etc. Overall, we end up with only a few dozen ranges. We now build a Huffman code that only specifies in which range the encoded value falls, and add $\log_2 s$ extra bits, where s is the size of the range, to specify the exact value in the range. This results in much smaller Huffman tables that can be stored for each image, while giving almost the same compression as a complete Huffman table. We then store the Huffman table itself, in canonical form, with the image data.

For ANS coding, we also code ranges rather than exact values, followed by extra bits, and store other types of meta information for decoding (instead of the code table in Huffman coding) with each image. As in Arithmetic coding, in ANS the number of bits for each symbol is amortized among a sequence of encoded symbols, which means symbols do not get assigned a direct code that has an integer number of bits. To apply ANS coding for the above framework, we first encode all the ranges using ANS, followed by all the extra bits for the image.

E. Experimental Results

In Table III, we show the bit rates (bits an integer takes on average) of coding the forward index with our Huffman coding approach, for different reorderings. Of course, whenever we sort by one field, then coding for that field improves. However, when sorting by codeword and by angle, we only exploit this effect; when sorting by X, Y, and size, we also enable duplicate suppression and exploit cases where a run of different Y values occurs with the same X. (Duplicate suppression is not that useful for codewords and angles, where there are few duplicates.) As a result, sorting by X, Y, and size outperforms the other two schemes, which have almost the same performance. Thus, we choose this ordering, together with the remapping proposed in Subsection 5.A.

In Table IV, we compare the bit rates of various other coding methods for the different features, when key points are sorted by X-Y-size. We show the results for the following coding techniques: Variable-Byte (Vbyte) [32], Simple9 [33], Golomb coding [31], OptPFD [20], Gzip, and our versions of Huffman and ANS coding[1] [10]. All methods compress best on the X coordinate, since X has a low entropy, as shown in Table I, and we are able to apply delta encoding for X. Angle is the most difficult feature to compress, since its entropy is 15.69, almost equal to 16 bits, which is the number of bits in a basic uncompressed representation. Thus, while there were visible spikes for angle along the four cardinal directions, as shown in Figure 3, most of the data lies outside the peaks of the spikes. Codeword is also hard to compress, while size and Y are relatively easy to compress. Among all the methods, Simple9 has the worst performance, and the reason is that for codewords and angles, it usually only packs a single value into each 32-bit word. (Note that Simple9 might fare better if we had sorted by codewords or angles.). Vbyte does not benefit as much as other approaches for X, since Vbyte does not do well when many numbers are much smaller than 2^7, and there are many small X values. Golomb coding performs the best for X, because it compresses small values well. Overall, Huffman and ANS outperform all other methods, indicating that for our forward index compression problem, integer coding techniques introduced in the context of compressing inverted indexes are not that useful. Huffman performs slightly better than ANS because the meta information stored under ANS is larger than the Huffman code table; this might change if images were much larger.

[1]ANS coding was implemented using the code at https://github.com/Cyan4973/FiniteStateEntropy

Features	by codeword	by angle	by X-Y-size
X	8.286	8.286	1.684
Y	8.412	8.412	6.861
size	9.592	9.592	8.155
angle	15.857	8.201	15.857
codeword	13.115	20.761	20.761
all	55.262	55.252	53.318

TABLE III: Bit rate under Huffman coding, for different features and different orderings.

Compressor	X	Y	size	angle	c.w.	all
None	16.000	16.000	16.000	16.000	32.000	96.000
Simple9	2.059	10.668	13.670	30.418	31.890	88.705
Vbyte	7.996	12.471	15.671	21.948	23.928	82.014
Gzip	1.884	10.638	11.869	15.984	22.835	63.210
Golomb	1.646	8.959	10.892	16.967	21.958	60.422
OptPFD	1.885	8.806	11.470	16.478	21.641	60.280
ANS	1.662	7.286	8.672	15.842	21.034	54.496
Huff	1.684	6.861	8.155	15.857	20.761	53.318

TABLE IV: Bit rates for different integer coding techniques.

For an image query, about 50% of the time is spent on feature extraction, 25% on object detection, and 25% on cascaded ranking in our system. In Table 5, we show the bit rate per key point, decompression cost per image retrieved from the forward index, and overall query execution cost in milliseconds. Since our main focus is to reduce the memory size of the forward index, we take ANS and Huffman coding and compare their performance. In our experiments, ANS took about twice the time to decode the features of an image than Huffman coding. Note that using Huffman or ANS in fact results in slightly faster overall query execution than using no compression, most likely because the smaller forward index results in better memory access and cache behavior. (The uncompressed forward index was also completely in main

memory.) However, the speedup is fairly small, and the main motivation for compression is the reduction in size by almost a factor of 2, meaning that larger indexes can be completely held in a given amount of main memory.

Compressor	bit rate	decompress cost (us)	image-query cost (ms)
None	96.000	N/A	242.6
ANS	54.496	391.31	239.8
Huff	53.318	198.64	232.1

TABLE V: Bit rate, decompression cost in us per image, and average query execution cost in ms, for ANS and Huffman coding.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we provide what we believe to be the first study of forward index compression in a real-world instance retrieval system for augmented reality applications. We compare various index compression approaches, and find that entropy-based coders such as Huffman and ANS result in the best compression ratio for our scenario, achieving compression of about a factor of two, while also very slightly speeding up query processing. Our implementation is tested and deployed in Blippar's AR platform.

While our approach is a promising first step, we suspect there might be additional patterns in the data, say between similar images, that could be explored in the future for better compression. Furthermore, when additional features are involved, our findings could still be useful and other compression ideas might apply.

We also plan to further design suitable lossy compression schemes and explore the trade-off between retrieval quality and forward index size, and we believe we might be able to achieve significant reduction of index size with minor or even no quality loss.

REFERENCES

[1] L. Zheng and Y. Yang and Q. Tian, "SIFT Meets CNN: A Decade Survey of Instance Retrieval," in IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017.

[2] D. G. Lowe, "Object recognition from local scale-invariant features," in Proceedings of the 7th IEEE International Conference on Computer Vision, 1999.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems 25, 2012.

[4] H. Houdong, W. Yan, Y. Linjun, K. Pavel, H. Li, C. Xi, H. Jiapei, W. Ye, M.Meenaz, and S. Arun, "Web-Scale Responsive Visual Search at Bing," in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining, 2018.

[5] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. "Object retrieval with large vocabularies and fast spatial matching," in IEEE Conference on Computer Vision and Pattern Recognition, 2007.

[6] M. Siedlaczek, Q. Wang, Y. Chen and T. Suel, "Fast Bag-Of-Words Candidate Selection in Content-Based Instance Retrieval Systems," in IEEE International Conference on Big Data (Big Data), 2018.

[7] L. Wang, J. Lin, and D. Metzler, "A cascade ranking model for efficient ranked retrieval," in Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval, 2011.

[8] Q. Wang, C. Dimopoulos, and T. Suel, "Fast First-Phase Candidate Generationfor Cascading Rankers," in Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, 2016.

[9] J. Duda, "Asymmetric numeral systems as close to capacity low state entropycoders," in arXiv:1311.2540v2, 2014.

[10] A. Moffat and M. Petri, "ANS-Based Index Compression," in Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 2017.

[11] H. Jgou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in European Conference on Computer Vision, 2008.

[12] Y. Zhang, Z. Jia, and T. Chen,"Image retrieval with geometrypreserving visual phrases," in IEEE Conference on Computer Vision and Pattern Recognition, 2011.

[13] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman,"Object retrieval with large vocabularies and fast spatial matching," in IEEE Conference on Computer Vision and Pattern Recognition, 2007.

[14] H. Jgou and A. Zisserman, "Triangulation embedding and democratic aggregation for image search," in IEEE Conference on Computer Vision and Pattern Recognition, 2014.

[15] H. Jgou, M. Douze, and C. Schmid,"Hamming embedding and weak geometric consistency for large scale image search," in European Conference on Computer Vision, 2008.

[16] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in IEEE Conference on Computer Vision and Pattern Recognition, 2006.

[17] J. Zobel and A. Moffat, "Inverted files for text search engines," in ACM Computing Surveys, 2006.

[18] D. Blandford and G. Blelloch, "Index compression through document re-ordering," in Proceedings of the Data Compression Conference, 2002.

[19] F. Silvestri, R. Perego, and S. Orlando, "Assigning document identifiers to enhance compressibility of web search engines indexes," in Proceedings of the ACM Symposium on Applied Computing, 2004.

[20] H. Yan, S. Ding, and T. Suel, "Inverted index compression and query processing with optimized document ordering," in Proceedings of the 18th International Conference on World Wide Web, 2009.

[21] J. Dimond, P. Sanders, "Faster Exact Search using Document Clustering," in International Symposium on String Processing and Information Retrieval, 2015.

[22] Q. Wang, T. Suel, "Document reordering for faster intersection," in Proceedings of the VLDB Endowment, 2019.

[23] P. Boldi, M. Santini, and S. Vigna, "Permuting web and social graphs," in Internet Mathematics, 2009.

[24] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P.Raghavan, "On compressing social networks," in Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009.

[25] K. H. Randall, R. Stata, R. G. Wickremesinghe, and J. L. Wiener, "The link database: fast access to graphs of the Web," in Proceedings of the Data Compression Conference, 2002.

[26] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, A. Shalita "Compressing Graphs and Indexes with Recursive Graph Bisection," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

[27] M. Poess and D. Potapov, "Data compression in oracle," in Proceedings of the 29th International Conference on Very Large Data Bases, 2003.

[28] D. Lemire and O. Kaser, "Reordering columns for smaller indexes," in Information SciencesInformatics and Computer Science, Intelligent Systems, Applications: An International Journal, 2011.

[29] D. J. Abadi, S. R. Madden, and N. Hachem, "Column-stores vs. row-stores:How different are they really?" in Proceedings of the ACM SIGMOD International Conference on Management of Data, 2008.

[30] A. L. Holloway and D. J. DeWitt, "Read-optimized databases, in depth," in Proceedings of the VLDB Endowment, 2008.

[31] I. H. Witten, T. C. Bell, and A. Moffat, "Managing gigabytes: compressing and indexing documents and images" in IEEE Transactions on Information Theory, 1995.

[32] F. Scholer, H. E. Williams, J. Yiannis, and J. Zobel, "Compression of inverted indexes for fast query evaluation," in Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, 2002.

[33] V. N. Anh and A. Moffat, "Inverted index compression using word-aligned binary codes," in Information Retrieval, 2005.

[34] C. D. Manning, P. Raghavan, and H. Schtze, "Introduction to information retrieval," Cambridge University Press, 2008.