SketchSeeker: Finding Similar Sketches

Seth Polsley, Jaideep Ray, and Tracy Hammond

***Abstract*—Searching is a necessary tool for managing and navigating the massive amounts of data available in today's information age. While new searching methods have become increasingly popular and reliable in recent years, such as image-based searching, these methods may be more limited than text-based means in that they do not allow generic user input. Sketch-based searching is a method that allows users to draw generic search queries and return similar drawn images, giving more user control over their search content. In this paper, we present SketchSeeker, a system for indexing and searching across a large number of sketches quickly based on their similarity. SketchSeeker introduces a technique for indexing sketches in extremely compressed representations, which allows for fast, accurate retrieval augmented with a multilevel ranking subsystem. SketchSeeker was tested on a large set of sketches against existing sketch similarity metrics, and it shows significant improvements in terms of storage requirements, speed, and accuracy.**

***Index Terms*—Approximate sketch matching, computer vision, databases, sketch descriptors, sketch retrieval.**

I. Introduction

SKETCHING is a universal form of expression. Humans can render any object on any kind of surface using sketching, and with touch devices becoming an increasingly integral form of communication, it is important that sketch-based systems be applied to more domains. Searching is one domain where sketching is a relatively new form of input. The amount of data to search through increases every day, and with users wishing to search in more and more modalities, sketch-based searching could offer a fast and flexible solution using simple drawings like those seen in Fig. 1.

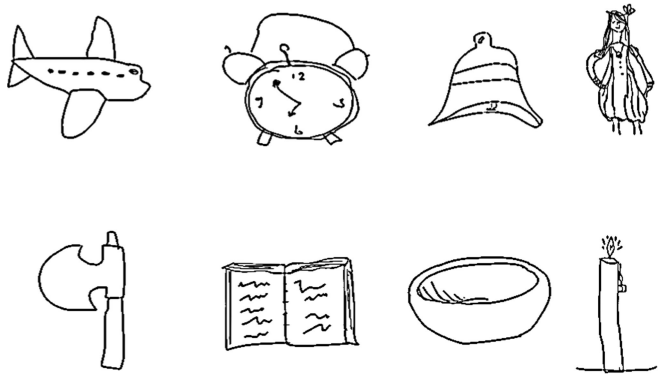

Fig. 1. Human-drawn object sketches.

However, efficient searching requires small indices that can deliver speed and accuracy. While much exploration has been done toward efficient image-based searching in recent years, sketch retrieval is a relatively new field that has widespread applications. One example could be searching for clip art based on a sketch. In this paper, we describe SketchSeeker, a sketch retrieval engine that combines state-of-the-art techniques to achieve fast, accurate, and ranked search results. SketchSeeker introduces a method for building a sketch search index that is extremely compact but still sufficiently representative of the full content to achieve high accuracy. Furthermore, it uses these "sketch signatures" to perform rapid searches over a large number of sketches that are boosted by object labeling to provide some semantic-oriented retrieval. Because SketchSeeker is a modular system that combines multiple techniques, it has two significant contributions to the field of sketch retrieval: 1) enormously-compressed, representative "sketch signature" embeddings for index searching and 2) a components-based framework for retrieval that allows extensible stages like ranking, which we use to implement an interpretation of semantics in this paper to boost performance.

SketchSeeker is a single system that combines three distinct subsystems. The first such subsystem is the sketch-indexing stage. In this stage, we describe a method for optimizing storage of sketches in a database using a highly-compressed, searchable representation. The sketch descriptors, which consider both shape and structure, along with the compression performed using a deep autoencoder architecture, improve the retrieval in terms of time and space complexity when compared to other known methods. The second subsystem is query retrieval. Similar sketches are returned based on their shape and structure according to their distance from the query shape in a kD-tree, as bound by an empirical threshold. The final component is a ranker, which sorts retrieved sketches based on two layers of filtering. First, semantic filtering is performed on the search results using a support vector machine (SVM) classifier that returns the most likely label for each sketch. After the filtering based on most likely meaning is complete, a median filter is used to eliminate any outliers before returning the final result set.

The remainder of this paper is laid out as follows. In Section II, we discuss the related works to SketchSeeker. Since sketch retrieval is similar to many other searching problems, and this solution relies upon many different components for portions such as sketch descriptors, semantic filtering, and ranking, there are a multitude of related works from varying domains. An additional section on sketch similarity itself appears in Section III in order to motive SketchSeeker more clearly and set it

Manuscript received November 16, 2015; revised June 18, 2016 and December 2, 2016; accepted December 19, 2016. This paper was recommended by Guest Editor E. Anquetil.

The authors are with the Sketch Recognition Lab, Texas A&M University, College Station, TX 77843 USA (e-mail: spolsley@tamu.edu; jaideep.ray@tamu.edu; hammond@tamu.edu).

Digital Object Identifier 10.1109/THMS.2017.2649684

apart from existing methods. Sections IV and V contain a more detailed description of the system and its components, divided into the three primary subsystems previously discussed. Results and discussion are then provided in Section VI to compare SketchSeeker against other similarity metrics and evaluate its retrieval performance. The paper closes by considering future work and conclusions in Sections VII and VIII.

II. Related Works

Sketch retrieval overlaps with a number of other areas. In addition to the connection with searching and classification, it relies heavily on sketch compression and representation, as well as similarity metrics. Most previous work in the area of sketch searching has been based on approximate matching using a sketch descriptor. The focus of these papers was the accuracy of the descriptor in capturing sketch features. To name just a few, these descriptors include shape context, scale-invariant feature transform (SIFT) descriptors, Hausdorff metric, and Fisher vectors. The next section provides a more directed discussion of similarity metrics and shape descriptors, but much of the relevant work related to sketch recognition, classification, categorization, and retrieval is mentioned below.

A. Sketch Recognition

Sketch Recognition is the algorithmic recognition of hand-drawn sketches. It consists of many different algorithms, some designed for specific domains and others designed for general recognition. It is primarily feature-based recognition, but features extracted from a sketch may be visually, geometrically, or modality based. Rubine [1] features are the most well-known geometric sketch features. Rubine features have been extended for sketch recognition algorithms in multiple domains. The $1 recognizer can detect multiple geometric shapes using corner detection algorithms and Rubine features [2]. Topological information about sketches have also been used for recognition as well. This approach works well with sketches having multiple spatial components like chemical bonds or complex electrical circuits [3]. Ladder [4] is a language to describe how sketched diagrams in a domain are drawn, displayed, and how recognition algorithms can be written for that domain [4].

B. Template Matching

Template matching aims at finding similarities between a given sketched figure and the query. While template matching has very broad applications, in sketch-related domains, it has primarily been applied to geometrical shapes and engineering drawings [5]–[7]. While template matching can be very effective at identifying if two sketches are similar geometrically, it is important to remember that human object sketching is highly varied. As a result, template matching methods that work well with geometric shapes may fail in the case of free hand sketches.

C. Classification for Sketching

Multiple techniques for image classification have been applied to hand-drawn sketches with some success. SIFT, Fisher vectors, and such descriptors have been used for sketch classification [8]–[12]. SIFT and Fisher vectors are techniques for encoding an image according to its gradient. Blocks of the image are described by the direction of change relative to the cardinal and intermediate directions, eight orientations in total. There have also been some attempts to do a classification-driven analysis that can predict the semantic aspects of sketch [13].

D. Multimodal Search

Here, multimodal search refers to searching by sketching across different domains. The sketched lines are taken as the input query and used to retrieve media files, images, or three-dimensional shapes [14], [15]. While there has been some interest in multimodal searching, sketch retrieval has not been as explored [16].

E. Sketch Retrieval

Sketch-to-sketch retrieval is a known problem but has not been well-explored. The popularity of stylus-based and touch devices have made sketching an important means of digital communication, further emphasizing the importance of sketch understanding in domains like search. Sketch matching and retrieval has various applications, such as trademark logo matching [17], handwriting recognition, or general searching on sketch-based interfaces or general stylus-based surfaces. Sketch retrieval according to shape topic has been employed in MindFinder [18], [19].

F. Sketch Representation

Also relevant to sketch retrieval is the representation method of a sketch. Sketches can be very large to store and search through when considering all of the raw point data, so for retrieval, it is important to have a representation that balances compression with the complexity needed to accurately represent a sketch. Methods such as SIFT descriptors [20] and Fisher vectors [13] are forms of representing sketches according to different kernels. Further, shape context descriptors, introduced in [21], have been shown to capture the shape information of a sketch effectively. While all of these generate a searchable representation of sketches, such descriptors are huge matrices, and it is difficult to work with them in their original form. An approach to compress shape context descriptors for efficient shape matching has been discussed in [21]. In [22], Oltmans introduces a new vision-inspired feature for representing sketches. This "Bullseye" feature uses a sliding window algorithm to generate a circular histogram of an image that may be used for match finding.

III. Sketch Similarity

A sketch retrieval system must have an indexing and a matching framework. Indexing involves generating a compressed sketch representation, whereas matching relies on some similarity metric to find the nearest sketches. In this section, we provide a more detailed discussion of some of the indexing and matching methods relevant to SketchSeeker.

Ideally, the indexing framework of a retrieval system will allow for a very compressed representation of a sketch, while still maintaining the ability to effectively match. The matching framework defines similarity between two sketches and should be symmetric

$$\text{dist}(A, B) = \text{dist}(B, A)$$
$$\text{dist}(A, A) = 0.$$

A. Hausdorff Distance

One common method for evaluating how well sketch *A* matches sketch *B* is Hausdorff distance [23], a cost metric defined by the following two equations:

$$h(A, B) = \max_{a \in A}(\min_{b \in B}(|a - b|))$$
$$H_d(A, B) = \max(h(A, B), h(B, A)).$$

Hausdorff distance measures how far the shapes are from being isometric. It considers each sketch as a cloud of points, which we achieve by obtaining about 100 samples from the sketched drawings randomly. Unfortunately, Hausdorff is a pairwise match evaluator, so to use it within a retrieval system's matching framework, it must scan through all the sketches to find the best match. This is computationally expensive, time consuming, and inefficient in terms of storage. Furthermore, the original Hausdorff algorithm considers only the maximum distance between points. As Ma and Doermann point out, this makes it extremely sensitive to a single high-distance outlier; while their generalized method improves on this limitation by selecting a kth distance point, they still found it to be highly dependent on the data [23]. This is an important factor since we are dealing with user-drawn sketches of natural objects.

B. Feature Descriptors

A more feasible approach to indexing and matching is based on feature descriptors. Such descriptors are widely used in computer vision problems and image matching, so we should consider them when matching sketches. Feature descriptors encode certain features about a sketch like topological information of the sketch (connectedness amongst components), geometry (shape), or local features like corners or curvature.

It is a challenging task to detect what kinds of features will work best for matching sketches. Certain feature descriptors are invariant to scale, rotation, or affine transformations and may work really well for matching human sketches of everyday objects. The key issues are detecting the feature points, encoding them effectively, and matching them, which may not correspond directly with techniques used for images. Much work has been done recently on content-based image retrieval [24], but images have rich description, color, texture, and shape, whereas sketches have shape, temporal, and spatial stroke information only. We use adapted feature descriptors for SketchSeeker.

While there are multiple types of feature descriptors for images, we use geometric feature descriptors in SketchSeeker. Feature descriptors can also be classified according to whether they are global or local (also called "structural"), pertaining to whether they describe attributes of a complete sketch or only a localized region [25].

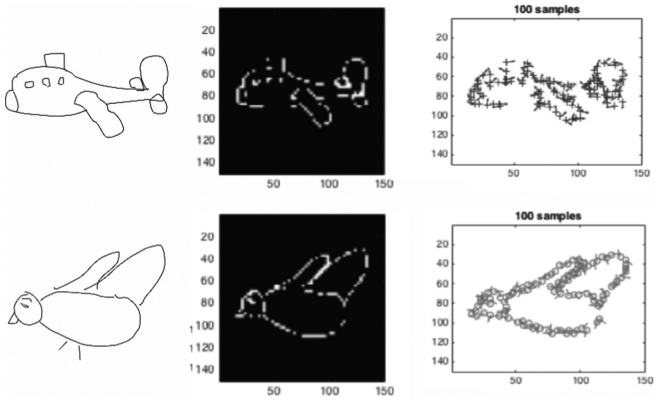

Fig. 2. Shape context descriptors extraction showing how a sketch is sampled into a cloud of points before feature extraction.

1) Shape Context Descriptors: Shape context is a geometric, global feature descriptor. Shapes are very discriminative features of sketches, making this a useful descriptor in terms of indexing and matching sketches. The intuition of shape contexts is based on histograms of radial-bins [21], [22]. In more detail, a circular window is placed over a sketch which is divided into bins by radial lines and concentric circles [26]. Then, for every point p_i, a measure of relative distance is created based on the following histogram equation:

$$h_i(k) = \#\{q \neq p_i : (q - p_i) \in \text{bin}(k)\}.$$

This defines the shape context for p_i so that the resulting set of feature descriptors is an array of shape contexts for the points in the sketch. This method of construction is largely resistant to noise and outliers.

SketchSeeker uses shape context descriptors as just one component of sketch matching. These feature descriptors can consume a lot of space when stored for a large number of sketches, which the original authors addressed through a combination of clustering and histograms to construct "shapemes." SketchSeeker forgoes shapemes, using instead an embedded representation of the descriptors generated by an autoencoder. Fig. 2 shows the shape context description of two sketches, and Figs. 3 and 4 show the calculated costs and similarity of matching the sketches.

2) SIFT Feature Descriptors: For reliable description or for tasks like recognition or retrieval, it is important that the features extracted from an input are invariant to scale change, small noise, or small affine transformations. SIFT is one such algorithm from computer vision that has been widely used for object recognition [27]. SIFT is geometric and local; there are two components to its implementation—keypoint detector and descriptor. These keypoints are regions of the image with a high rate of change, like corners. Essentially, in our case, SIFT is a histogram of image gradients on a sketch.

One of the successful uses of SIFT features for sketch recognition has been done by Eitz *et al.* [9]. They extracted SIFT

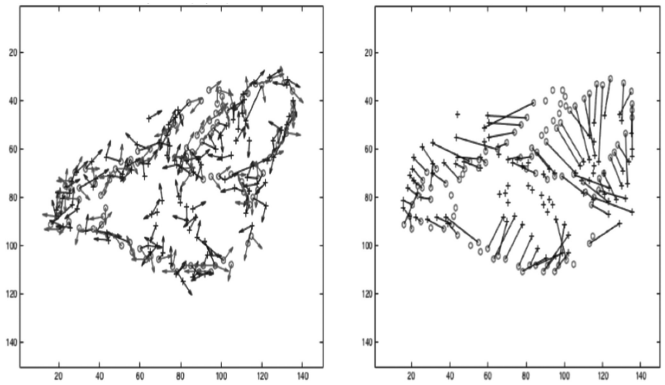

Fig. 3. Shape context matching cost as described in [26] with 80 descriptors; the lefthand image is warped, and the righthand unwarped.

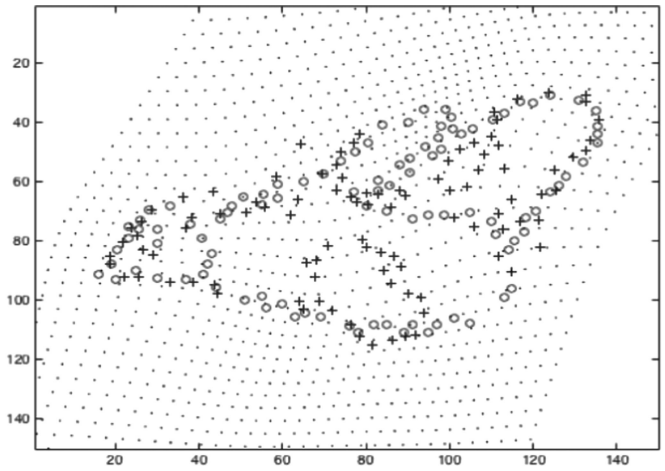

Fig. 4. Sketch similarity by shape context matching as described in [26] with an affine transform cost of 0.43859 and shape context cost of 0.14852.

features from sketches and trained an SVM classifier to categorize sketches. SketchSeeker has a similar approach, using SIFT as one portion of the indexing stage and an SVM classifier for semantic interpretation during the ranking stage. SIFT is highly compressed for SketchSeeker and paired with shape context to give more information about the input sketch; by splitting SVM classification into a single stage of the ranker, ranking can be performed considering other metrics.

IV. Methodology

A. Problem Formulation

Given an input query sketch, we must retrieve a collection of very similar sketches from the database in terms of shape and semantics. SketchSeeker's focus is to achieve extremely fast searches with the most storage-efficient indexing possible.

B. Sketch Dataset

SketchSeeker uses a dataset consisting of 20 000 human-drawn sketches of everyday objects. The dataset consists of 250 classes each having 80 sketches. As seen in Fig. 5, there is some ambiguity introduced by the similarity of the objects

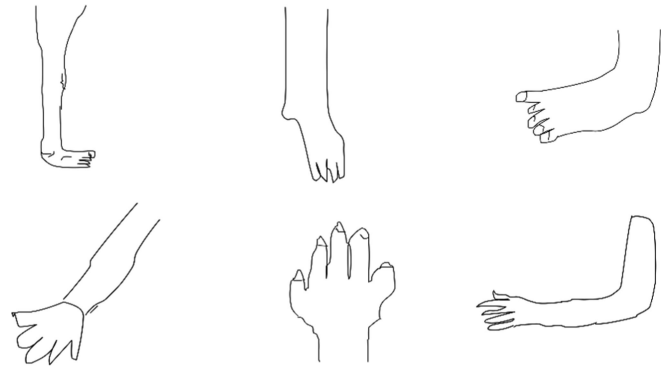

Fig. 5. Interclass similarity on legs and arms. Top—Legs and feet. Bottom—Arms and hands.

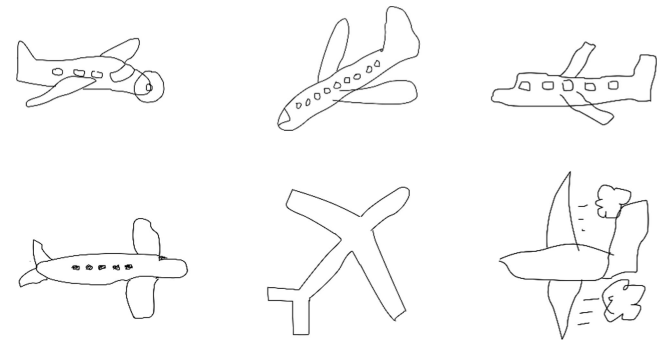

Fig. 6. Intraclass variation on airplanes.

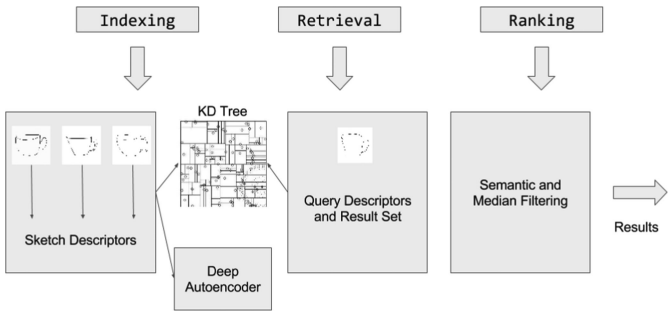

Fig. 7. SketchSeeker design overview.

being represented, referred to as interclass similarity. Futhermore, humans draw diversely while drawing the same object, which causes intraclass variation; see Fig. 6 for an examples with the airplane class. Sketches can also be noisy. We use a dataset of 20 000 sketches to capture as much variance, similarity, and noise as possible so that SketchSeeker can be made invariant to scale and noise while still maintaining discriminative capabilities between objects.

V. Implementation Details

SketchSeeker introduces a "sketch signature," a minimized hash representation of a sketch that captures both structure and shape. Using this minimized representation, it allows for fast searching across large datasets that can then be ordered and ranked to improve performance, as well as incorporate some

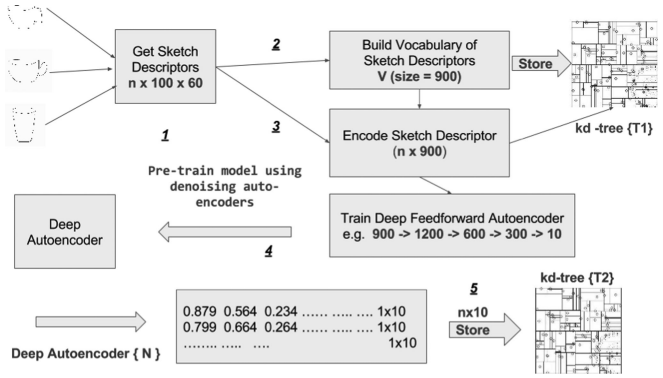

Fig. 8. Indexing overview.

semantics. The following sections will detail each of the steps in the general work flow and provide a more clear explanation of SketchSeeker's sketch signatures and other features.

As a high-level overview, SketchSeeker accepts an input query and returns a set of nearest similar sketches from the database. It splits the task into three primary components, as shown in Fig. 7—indexing, retrieval, and ranking. First, sketch images are indexed in the database. This process involves computing feature descriptors and substantially compressing the representation for storage. Second, retrieval is performed by finding the nearest neighbors to a query sketch in a searchable, multidimensional map of the sketches in the form of a kD-tree. Third, the ranker considers the results based on their shape context and SIFT matches, along with the object label determined by an SVM classifier. Finally, the ranked result set is provided as output to the user.

A. Indexing

1) Introduction: One of SketchSeeker's core contributions is its ability to compute exceedingly compact embeddings to represent a sketch that incorporate shape and structure; these are also referred to as sketch signatures. Providing heavy compression while still maintaining enough complexity to be searched effectively is one of the most significant aspects of indexing. Compression speeds up the matching process, yielding faster queries, and saves storage space, but because we try to match the sketch signature of a query sketch to the sketch signatures of sketches in the database, severe compression can reduce accuracy. The similarity between two such codes or sketch signatures should represent the similarity between sketch descriptors of the corresponding sketches. SketchSeeker combines both shape context and SIFT information into the signature, which is compacted with minimal data loss by a deep autoencoder. Fig. 8 shows a graphical overview.

2) Feature Extraction: A sketch is made up of multiple strokes. To describe a sketch completely we should consider both its appearance, e.g., shape, and the important stroke features that are captured in key-points like corners. SketchSeeker uses shape context descriptors to represent the overall shape of a sketch. To represent stroke features, we use SIFT descriptors. More specifically, because shape context is intended to capture shape, we refer to these features as "global descriptors," and because SIFT is better at representing stroke-level features, we refer to these features as "local descriptors." Thus, by working with both descriptors, SketchSeeker represents sketches as a combination of external shape and internal structure, enabling better distinction among complex sketches. During the indexing process, these features are extracted from each sketch in the database, which can be done in parallel to boost efficiency of the system.

3) Preprocessing: We preprocess the sketch descriptors by vector quantization. We first sample the sketches to a cloud of 100 points, a selection made to balance simple and complex sketches with a small number of points that can capture dense strokes. Shape context descriptors are 60 dimensional vectors, and since each sketch has 100 such descriptors, it becomes a 100×60 matrix. SIFT descriptors are 128 dimensional vectors, and each sketch has a different number of SIFT key-points depending on the strokes. So, SIFT descriptors are (number of key-points)$\times$128 matrix. We consider around 5000 SIFT key-point descriptors as the candidate set to build a vocabulary, discussed in the next step. SIFT features are extracted using the VLfeat library [28] and selected in the order they are sampled. We vertically concatenate the descriptors from all the sketches to form a composite shape context matrix and SIFT context matrix; the resulting matrix has a recurrent structure due to the inherent order of the descriptor sampling.

4) Vocabulary Building and Binning: To compress the sketch descriptors, we build a separate vocabulary for each kind of descriptor and express the descriptors in terms of this vocabulary. This process involves clustering all the descriptors and forming a vocabulary of all chosen cluster centers. The next step involves assigning the descriptor to an index of the cluster center to which they belong; we use approximate nearest neighbor (ANN) *k*-means to cluster the descriptors. After vocabulary building, we need to encode the descriptors with respect to this vocabulary. In the case of SIFT, where the number of key-points varies from sketch to sketch, we need to bin the sketch descriptor to get uniform size representation.

The detailed algorithm to determine the descriptor vocabulary is as follows.

1) Sample s points from the sketch.
2) Determine the feature descriptors of the sketch using these s points. Each shape context descriptor is an n dimensional vector ($n = 60$). Each SIFT descriptor is an m dimensional vector ($m = 128$). Note that the original number of shape context descriptors and SIFT descriptors was not necessarily n and m. To make the descriptors have the same length for each sketch, we employed a tactic of vector quantization. This is a common method that works by clustering the descriptors according to their values (magnitude and direction) and then reassigning them into a vector of cluster labels. This groups, or more technically bins, the descriptors into vectors of length n and m. The sketch is then represented as a collection of all feature descriptors.

3) Now, we perform an ANN k-means clustering to determine cluster centers and cluster formations of the shape context vectors.
4) The number of cluster centers C determines the size of the vocabulary. The shape context vectors are assigned the index of the closest cluster center.
5) We now represent each sketch as a collection of shape context descriptors which are just labels of cluster centers (integers from $1, 2, \ldots, C$).
6) The shape context descriptor becomes just a single histogram of cluster label frequencies. These frequency histograms have length $|C|$.

We used a value of $C = 600$ to build a sufficiently large vocabulary for the many different object sketches, but the representation is too large to be efficient in space and time. Autoencoding, the stage described in the next section, is used to build a compressed representation of C which is far more efficient.

5) Autoencoding: We use a deep autoencoder from the dimensionality reduction toolbox [29] to learn compressed representations of the quantized shape contexts. A deep autoencoder is a composition of two symmetrical neural networks, where the first half represents the encoding layer and the second half represents the decoding layer. The layers are restricted Boltzmann machines. In our case, the input to the deep autoencoder is a C-length vector, $C = 600$ for SketchSeeker. A sample encoding layer would look like

$$600 \rightarrow 780 \rightarrow 450 \rightarrow 200 \rightarrow 10.$$

The autoencoder learns a vector of 10 numbers from a 600 length shape context descriptor. This vector is the encoded version of input. The second part is performing the opposite of this network and decodes the input from this encoded version of 10 numbers.

The advantage of using an autoencoder to generate sketch signatures is multifold. First, this method generates an optimal representation of the sketch. The neural networks in an autoencoder are designed to minimize the difference between the input and output objects; the representation in the smallest, middle layer will be of a minimum size with maximal ability to represent a sketch. Second, because the embedded layer is a minimal representation of a sketch, similar signatures may represent similar sketches, so searching using matching or nearest neighbor signatures is a feasible similarity computation. In contrast, methods like "shapemes," which are compressed representations of shape context descriptors, can be used to speed up searching through pruning, but the full similarity metric must still be computed on the original features [26]. Finally, being based on neural networks, autoencoding is a nonlinear method of dimensionality reduction, and as we saw with compression of geometric shapes in [30], this can be more accurate for the same compactness when compared with linear methods such as principle components analysis.

The detailed encoder steps are listed below.

1) The deep autoencoders compress the shape context descriptors (size C) into a vector of p floating numbers where $p << C$. We used $p = 10$ for SketchSeeker. We tested with the larger value of 16 but ultimately found that it did not have a significant benefit on the performance to be worth the exchange in storage space.
2) The deep neural network is stored to be used in the query retrieval stage.
3) After encoding, each sketch is represented by a vector of p floating point numbers. This representation serves as the signature/index to the sketch.
4) We store all such signatures in another kD-tree for efficient lookups of nearest neighbors.

The main idea behind learning these codes is that similar codes correspond to similar sketches.

6) Efficiency: The data structure used to store the signatures should be efficient in time and space for the retrieval of nearest neighbors to a given query. We use a kD-tree to store all the sketch signatures as it facilitates easily computing distances for nearest neighbors. Note that kD-trees are not the most efficient data structures in time since there will be many distances computed, but once indexing is performed in SketchSeeker, the sketches are sufficiently compact that larger representations or other distance comparisons like Hausdorff would be much slower.

In terms of timing complexity, indexing is essentially an offline process, so it does not affect real-time retrieval latencies. Still, indexing performance is vital for large sketch databases. The major steps contributing to time complexity are the vocabulary building and autoencoder training. The vocabulary building is dependent on the runtime of the ANN *k*-means clustering algorithm. It is also dependent on the vocabulary size and max number of iterations allowed for convergence, which is 600 in SketchSeeker's case.

Regarding storage space, we need space proportional to the number of sketches. Because each sketch is represented by 10 numbers, SketchSeeker makes it possible to store very large sketch databases completely in-memory. Without this level of compression, the database could require an enormous amount of storage space. For comparison, the Hausdorff metric must store all the stroke points. Shape context requires the number of stroke points multiplied by 60 descriptors per sketch, and as mentioned before, SIFT varies in size based on the number of key-points. Each key-point will require 128 numbers. Clearly, SketchSeeker's method makes significant gains in terms of storage efficiency through its multilayered indexing approach and autoencoder augmentation.

7) Sketch Clustering Visualization: Before final evaluation, we also performed a visual analysis of our sketch descriptors to ensure their validity. This analysis was motivated by a desire to show that the combination of shape and structure descriptors could be used to identify similar sketches and make distinctions among classes. We first plotted the descriptors on a 2-D plane using dimensionality reduction. The t-SNE [31] algorithm was used to reduce the descriptors to two dimensions which were then plotted and clustered into groups of sketches with similar feature values; t-SNE provides a numerical means of grouping clusters. We then visually examined some of these sketch clusters to investigate the similarity within and dissimilarity among different groups. This exercise was performed for both sketch descriptors, and Fig. 9 displays the results of clustering the cup

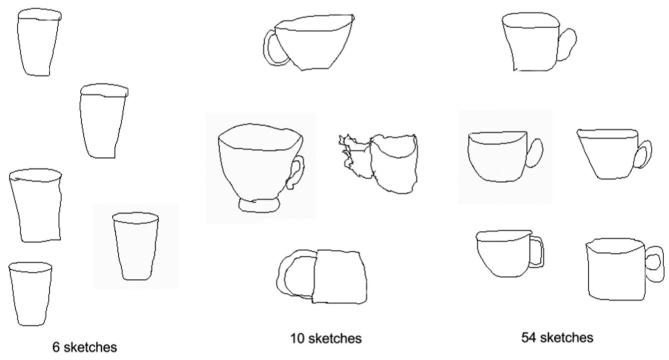

Fig. 9. Sketch clustering; a few representative sketches from each cluster are shown.

dataset. As we see, cups can be drawn in any number of ways (intraclass variation), so even within the class this variation may be extreme; this intraclass variation is one important motivation for selecting a large vocabulary as we discussed previously since it allows for "miniclasses" within larger classes. Likewise, we need separability among classes even when some features are shared (interclass similarity), and though we are only considering cups here, we can see that the clusters based on shape and structure have done a visually pleasing job of dividing within a class and show some potential for distinguishing among potentially similar sketches. Note that while there are 80 sketches to each class in the dataset, Fig. 9 is only showing a few of the most mutually distinct clusters of cups.

B. Query Formulation and Retrieval

1) Introduction: The next major stage of SketchSeeker is query retrieval. This requires indexing the queried sketch according to the indexing stage so that similar matches can be found in the kD-tree.

2) Preprocessing: We compute the shape context descriptor and SIFT descriptor for the query sketch. We use the same preprocessing steps to quantize the query sketch as before so that it becomes a sketch signature that may be compared with the stored sketches.

3) Query Formulation: We form two queries, one based on shape context descriptor and another based on SIFT descriptors. The first tries to seek out sketches that are similar in outer shape and the second one searches sketches that have similar internal structure. These two queries can be fired in parallel.

4) Retrieval: We use the queries to search for the nearest neighbors in the kD-tree containing all sketch signatures based on shape context descriptor and SIFT descriptor. Now, we have two sets of retrieved sketches based on the descriptors.

The detailed sequence of steps is as follows:

1) Extract the shape descriptors and SIFT descriptors from the input sketch and quantize it using the method mentioned in Section V-A. It will be a constant time one-time look-up operation since we already have the kD-tree with all cluster centers in memory.
2) Use the trained autoencoder to get the compressed representation of the quantized shape descriptor.
3) This compressed representation will serve as the signature of the sketch and our query.
4) We search for n nearest neighbors of the query sketch in the shape context kD-tree and SIFT kD-tree. The number n is defined empirically, and after testing with multiple values, we selected $n = 25$ as a good balance between result set size and tree sparsity. That is, the result set needs to be large enough for later steps like median filtering, but by setting n too high, many distant neighbors would be selected in sparse regions.
5) We use both these result sets as input to the ranker to form our final result set.

The nearest neighbors are sketches that are structurally similar to the query sketch as the descriptors cover both outer shape and internal key-points. But structure cannot be the only parameter for retrieving sketches. User sketches have a meaning associated with them. They may be sketches of commonly used objects, animate, or inanimate entities. In the next section, we explore the final stage of the pipeline and discuss how the results can be improved over the simple nearest neighbor comparison.

C. Ranker

1) Introduction: The ranker system boosts performance in multiple ways, one of the most significant being its semantic ranking. The semantic ranking is detailed more below, but from a high level, we are using this term to describe an understanding of the intention of the user and what she is trying to draw. In part this has already been captured by the autoencoders embeddings, since similar sketches will have related signatures, but we also use the object category of the query sketch so that results can be filtered according to related categories. This step also includes median filtering on the shape context and SIFT result sets based on matching distance to generate the final results. The ranking parameters for a result are given as follows:

1) semantic interpretation of query sketch;
2) shape context distance from query sketch s;
3) SIFT distance from query sketch d.

2) Semantic Filtering: We have tweaked the open-source Caltech 101 classifier for classifying sketches. The classifier uses dense SIFT descriptors and spatial histograms, and in our case, it was trained on the sketch dataset provided by TU Berlin. As mentioned previously, this dataset has 250 categories each having 80 sketches in them. We use 15 sketches per category to train the classifier and 15 for testing. The trained model is stored in memory to use during the ranking phase.

Because the SVM classifier can return likely object categories and scores, SketchSeeker can obtain a ranking of what the user might be intending to draw. In order for this to work, each sketch in the database is given a label, either manually, using the most common label of its nearest neighbors within a certain threshold, or its nearest neighbor's label if all neighbors have different classes. To perform semantic filtering, every user query is given a set of its top three labels using the SVM. SketchSeeker then filters the result set on these top three object categories by checking the label of each sketch in the result set, and if that

label does not match at least one of the top three output by the SVM, the sketch is discarded from the result set.

Predicting multiple labels instead of one improves our overall retrieval quality as it gives the classifier more flexibility. In early testing, just a single label was used, but the results often failed to capture the correct meaning of the query sketch, leading to poor results from the semantic filtering step. These poor results reflect the accuracy of the SVM classifier, which is about 65%. 65% is well over the random classification accuracy of 1/250 categories (0.4%), but it is insufficient for single-label filtering.

3) Median Filtering: In the final step of ranking, after the result set has been trimmed down using object categories, we examine the distances of the SIFT and shape context descriptors from the query sketch. These distances, which come directly from the distance between the sketch signatures in the kD-tree, are stored in the metadata of each sketch in the result set. The median distances for the SIFT results and the shape context results are selected. Then, any sketches that are beyond median distance of the original result set are filtered out; this trims the result set to a much closer group of nearest neighbors. Finally, these result sets are merged by a union operator that yields the final set of results presented to the user.

It is worth reiterating that the ranker stage is completely optional. Each SketchSeeker stage, beginning with the index-building, is extensible, and ranking can be boosted with any better global or domain-specific methods. Here, we have selected a simple SVM classifier trained on a relatively small dataset as one method of enhancing the retrieval through object labeling. The goal behind ranking with object label is to provide filtering based on user intention boosting the semantics-oriented output of the already similar content sketches from the kD-tree.

VI. Results and Discussion

Because SketchSeeker builds compact, fast sketch indices for searching, we have evaluated the system in terms of storage and speed efficiency alongside its final retrieval accuracy.

A. Storage Size

SketchSeeker's signature representation is extremely compact, allowing for enormous databases to be stored in-memory for fast querying. In contrast, other feature-based approaches do not scale as well since they generally store larger representations. On the extreme end, similarity metrics like Hausdorff must be computed against point clouds, so such matching systems must store sampled points for every sketch.

As shown in Table I, SketchSeeker requires only 10 numbers to save a sketch in the index, whereas other feature-based methods used in sketching tend to require much more storage. In Belongie *et al.* [26], their sketch index is comprised of $k = 100$ shapemes for each sketch. These shapemes themselves are compressed representations of the larger shape context features using histograms; SketchSeeker includes these same features even more compactly alongside the additional SIFT descriptors and the ability to easily compare based on embeddings nearest neighbors. Other similarity metrics based on point comparison would need to store samples for each point. A low-resolution sketch of only a hundred points already takes up significantly more space than either of the considered feature-based ones, but a high-resolution sketch that might be needed for good Hausdorff accuracy would require very space-consuming indices.

TABLE I
Comparison of the Sketch Index Size for a Few Different Storage Methods

Storage Size		
Method	Floats per Sketch	Search Index Size
SketchSeeker	10	12.8 MB
Shapeme	100	128 MB
Low-resolution sketch	200 (100 *x,y* pairs)	256 MB
High-resolution sketch	2000 (1000 *x,y* pairs)	2.56 GB

TABLE II
Latencies of Each Sketch Matching Metric Over All Sketches

Latency for 100 Queries			
Method	Median	Average	Std Dev
SketchSeeker	1.8 s	1.9 s	0.43 s
Hausdorff retrieval	192.5 s	195 s	16 s
Shape context	290.2 s	297 s	18 s

The resulting sizes shown in Table I are based on the selected dataset of 20 000 sketches, but if we consider larger datasets, it is easy to see how an order of magnitude makes a significant difference. For a dataset of 20 million sketches, SketchSeeker would need only a 12.8 GB index, which may be stored entirely in-memory today even on many desktop computers, while a 128 GB index for shape contexts could only be supported on high-memory servers.

B. Query Latency

In addition to small search indices, a sketch retrieval system working in real-time must have a low-latency sketch matching system. The basic unit of a sketch matching system is the average time taken to compute similarity between two sketches since this impacts how long a user of such a matching system has to wait to receive results. We consider the latency of our system against those for three of the most popular matching methods used today: Hausdorff distance comparison, shape context matching, and SIFT descriptor matching.

Table II shows the average matching latency for each of the methods tested. The latency was computed as the average time taken to complete over a total of 100 sketch queries, including both simple and complex shapes. Because sketch signatures are stored in kD-trees, SketchSeeker can match against all nearby sketches in the database in a single query; the other metrics use indices as dictionaries upon which to search through all entries for close matches, leading to dramatically slower query times.

Since SketchSeeker is designed to check against an entire collection of sketches easily, it is not surprising that it could complete each of the sample queries so quickly. A more interesting metric is looking at the latency of SketchSeeker for a

TABLE III
LATENCIES OF EACH SKETCH MATCHING METRIC ONLY CONSIDERING A SINGLE COMPARISON

Latency of Comparing Query with a Single Sketch			
Method	Median	Average	Std Dev
Hausdorff Retrieval	502 ms	514 ms	15 ms
Shape Context	913 ms	923 ms	115 ms
SIFT Descriptors	80 ms	83 ms	6 ms

TABLE IV
PRECISION OF THE RESULT SET FOR MULTIPLE METRICS, INCLUDING SKETCHSEEKER

Precision			
Method	Median	Average	Std Dev
SketchSeeker	87.5%	80%	25%
Shape context with embeddings	85%	74%	30%
Hausdorff retrieval	20%	25%	18%

complete database search against a single matching operation in the other metrics. Again, we compute the latency for the three other popular matching methods, but we only consider the time taken for determining one match. Table III shows that each of the other similarity measures—Hausdorff distance, SIFT descriptor distance, and shape context matching—need tens to hundreds of milliseconds for a single comparison. SketchSeeker has no concept of a single pairwise comparison, but as we saw in Table II, it completes a match against the whole database of 20 000 sketches in just under 2 s.

C. Retrieval Accuracy

While SketchSeeker has certain advantages in compression and speed, it is also important to consider the accuracy of its results. We compare SketchSeeker retrievals against those generated by the Hausdorff distance metric and a form of shape context. To do this, we perform a query in SketchSeeker that will return a set of K nearest neighbors based on a threshold. We then use the same query sketch and scan through the entire database of sketches calculating the top K sketches based on Hausdorff distance and shape descriptors. This approximates the output of SketchSeeker by finding nearest neighbors using only the Hausdorff or shape descriptor metrics.

The precision is measured according to the following definition of a true positive.

1) All sketches in the result set are compared against the query sketch according to Hausdorff similarity. This ensures that results are precise in the sense of being traditionally similar sketches. Only those results within a certain similarity threshold are considered as true positives.
2) The class label of each result is compared against the three most probable labels for the query sketch. For this dataset, all labels have been generated manually, so we could compare only the top class label, but in order to provide greater flexibility in the role of semantic labeling, allowing the precision metric to weight visual similarity somewhat higher than just label matching, three classes have been considered.

All other sketches in the result set are considered as false positives.

Based on this definition of precision, we can construct a performance comparison among several methods. Note that this is not purely a classification problem, as we are measuring precision both in terms of similarity and class; from this perspective, other classification-centric measures are not the most appropriate. For instance, recall in this case does not provide a valuable measure since we are always returning a few top results in terms of similar nearest neighbors, and the number of the actual 80 sketches returned from the sketch class is not as relevant.

Table IV shows the findings for several methods. Let us first discuss the values for SketchSeeker against shape context with embeddings. This implementation of shape context is very similar to the original form introduced in [26], but rather than using shapeme compressed representations, SketchSeeker's autoencoder embeddings are used. This method saves time since shapemes can only be used for pruning but the full shape context descriptors are used in comparison. SketchSeeker's embeddings can find similar sketches simply by storing them in a kD-tree and retrieving nearest neighbors, providing the enormous performance boosts we saw in the query latency analysis. Not only is SketchSeeker's embeddings method faster, it also achieves higher average precision owing to the fact that it captures SIFT descriptors as well.

We also see from Table IV that Hausdorff performs poorly. There are a couple of reasons for this result. First, the other two methods employ ranking at the end stage and receive a performance boost through SVM labeling. Because we wished to include both similarity and class label in the precision measure, results that match classes achieve a higher score. If we were to run this ranking stage on the Hausdorff results, then its accuracy would be perfect because it would match the definition of our precision measure (Hausdorff similarity with class labels). Instead we included standalone Hausdorff as a means of testing the performance of a pure similarity measure without any sort of object understanding. To provide a more fair comparison, we also ran the same tests on SketchSeeker with the ranking stage disabled. In those tests, SketchSeeker demonstrated an average 9% performance drop (71% accuracy); thus, Hausdorff was still outperformed when run against a system that did not include the SVM labeling, which leads to the second point of discussion. The Hausdorff metric may work reasonably well on simple-shaped sketches, but the precision drastically goes down with complex sketches. In such sketches, a point cloud comparison may show two sketches to be quite similar, but the internal structure may actually be very different. Feature-based methods are more resilient to these issues, although SketchSeeker is the only one that includes both shape and structure in its representation.

From Table IV, we also noticed that the results suffer from a high standard deviation. This is likewise due to the varying complexity of sketches. These values were computed over 100 sample queries and the precision of each result set.

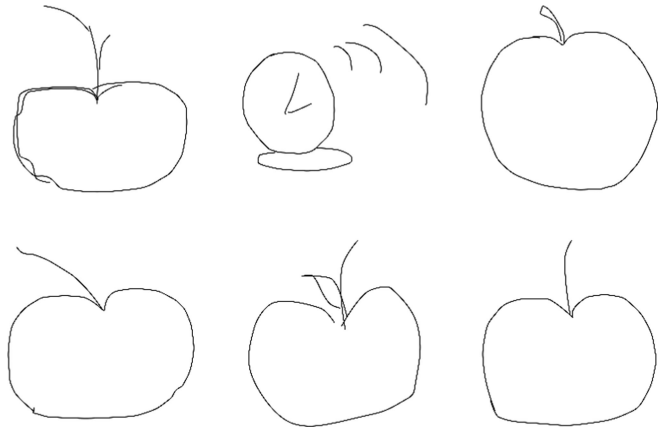

Fig. 10. Apple sketch matched by Hausdorff metric.

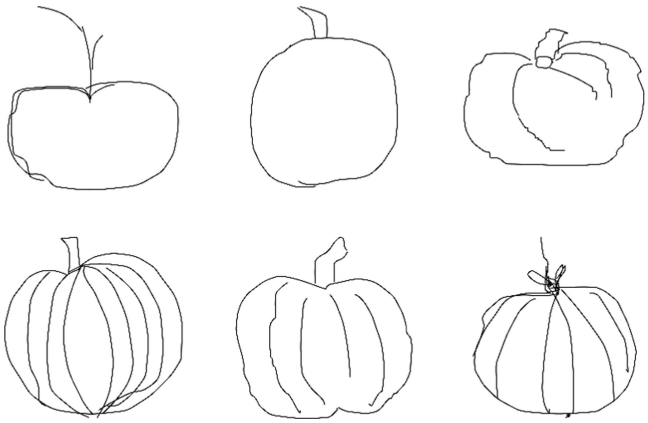

Fig. 11. Apple sketch query retrieval by SketchSeeker using shape context descriptor alone; note the false positives due to shape context descriptor matching of overtraced strokes.

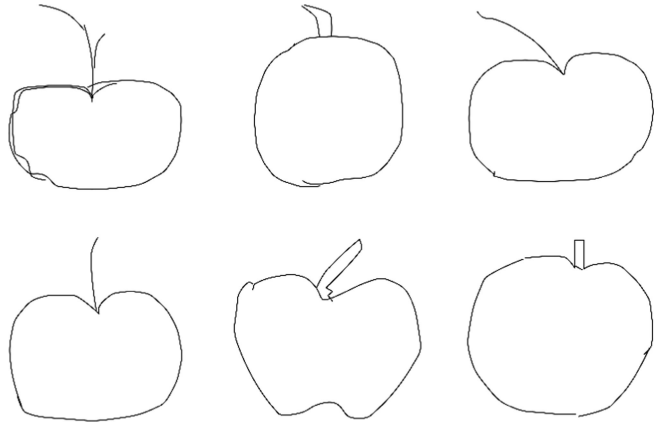

Fig. 12. Apple sketch query retrieval by SketchSeeker.

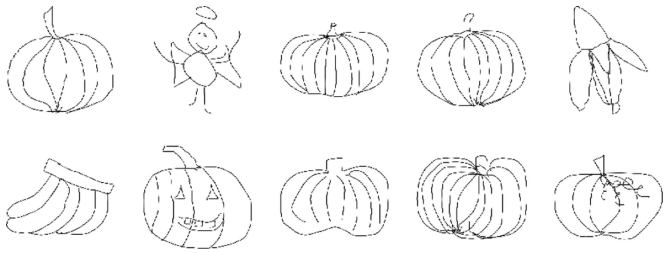

Fig. 13. Results of pumpkin retrieval by SketchSeeker based on shape descriptors alone; note the false positives due to inner structure of pumpkins.

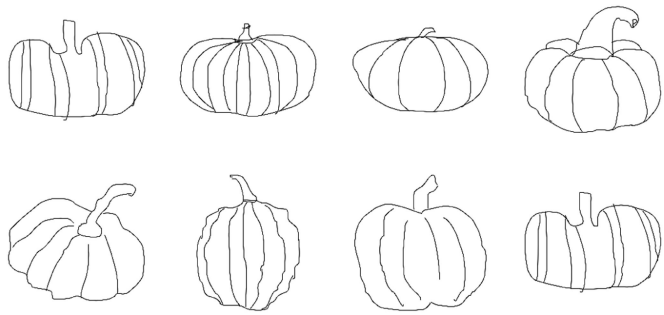

Fig. 14. Results of pumpkin retrieval by SketchSeeker.

Consistent with the entropy of the overall dataset, within those 100 queries was a mixture of simple, moderately complex, and very complex sketches. Ones that are very simply may need only outer shape to identify them, while others that are highly complex will have a lot of internal structure. However, we see that the median precision is higher than the average for SketchSeeker. This indicates that while complex sketches may account for a lot of variation and be difficult to match, most of the time, sketches will be midway between simple and complex, and in these cases, SketchSeeker performs quite well. This analysis of the standard deviation is only interesting in regards to precision; we did not see the effects of complexity in the latency evaluation since the searches require the same amount of time no matter the sketch content.

Further, consider the case of overtracing, which may often be the case for novice sketches. In Fig. 11, we see the sensitivity of the shape descriptor to overtracing. Even though the general shape is fairly accurate around the edges, the extra strokes lead to sketches with more complexity. When the shape is relatively simple with no overtracing, Hausdorff and shape context can be fairly accurate, although Hausdorff is not as sensitive to overtracing as seen in Fig. 10. SketchSeeker performs well regardless, being largely insensitive to certain limits of the other metrics such as scale, outliers, and overtracing; see Fig. 12. SketchSeeker also performs well on noisy sketches. Note that for comparison, in all query retrieval figures the first sketch shown is always the query sketch.

In sketches having a rich inner structure, both Hausdorff and shape context descriptors fare poorly against SketchSeeker's whole-sketch representation. See Figs. 13–16 for examples.

A complicated sketch category is "arm." It does not have a fixed shape and has features like muscles, fingers, and an elbow. Retrieval precision for "arm" sketches are low, but the SIFT query retrieval precision is high. Fig. 17 shows a query with an arm. The predicted labels for this category are "arm" and "foot," again reminding us of interclass similarity. This is a good

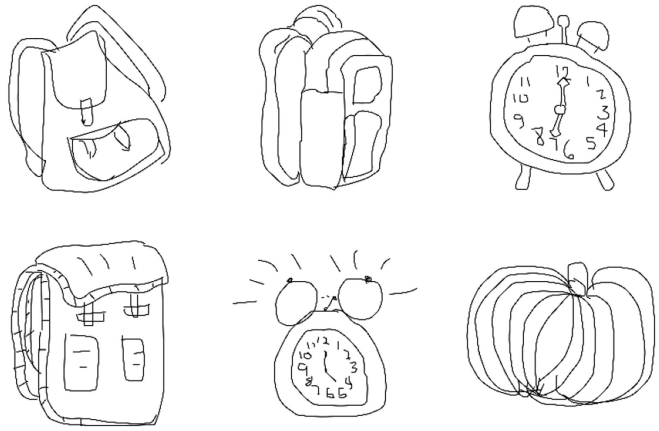

Fig. 15. Results of bag sketch retrieval based on uncompressed shape descriptor matching alone.

Fig. 16. Results of bag sketch query retrieval by SketchSeeker; compare the results with uncompressed shape descriptor matching in Fig. 15.

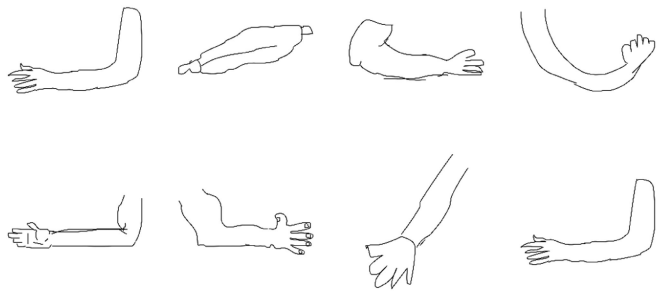

Fig. 17. Arms sketch retrieval by SketchSeeker.

illustration of using multiple features to handle the interclass and intraclass complications.

VII. Future Work

There are several interesting areas of research related to SketchSeeker that we would like to develop further. First, it would be worth exploring a more stable feature construction technique than concatenation of descriptors. Related is whether we can combine the global shape descriptor and local SIFT descriptor to come up with an entirely new descriptor for the index structure. Ideally, such a feature will include a measure of the sketch's entropy so that queries will be able to return sketches of approximate complexity as SketchSeeker tries to do using two features. The current indexing process is time consuming due to the long deep autoencoder training phase. We would like to investigate some parallel approaches to speed it up. Also, we would like to investigate giving sketches a better semantic interpretation than the one described in this paper, since the SVM classifier is somewhat limited in its capabilities. On a related note, because the sketch signatures group similar sketches, it would be interesting to explore how SketchSeeker's framework could be applied to recognition and classification problems. Finally, we would like to extend this system to a sketch search engine platform that could retrieve results for any generic user inputs and acquire new sketches to expand its index. In consideration of the broader field of sketch recognition, it is important to note that we have not explored the temporal information of a sketch's strokes in this paper. Temporal information is widely used in sketch recognition application, and classification based temporally on strokes might lead to a feature in the matching framework in the future.

VIII. Conclusion

In this paper, we described SketchSeeker, a sketch retrieval system for finding sketches similar to a given query sketch. We use both shape context and SIFT key-point descriptors in the matching framework. These sketch representations are then heavily compressed using deep autoencoding and stored in a kD-tree for enormous improvements in storage and speed efficiency. Finally, we rank the result set retrieved for an input sketch by the semantic interpretation of the query paired with median filtering on the distance of the matches to the query sketch.

Our approach toward sketch retrieval closely follows the design for any generic text or image retrieval system. The three subsystems described in our work—indexing, retrieval, and ranking—can be extended to design a domain-specific sketch recognition system that can be used for specific tasks such as logo/trademark retrieval, engineering drawing retrieval, clipart finding, or multimodal searching systems. Furthermore, the nearest neighbor search technique could be used to provide real-time suggestions for stroke completion to a user of a sketch-based interface. Overall, we show that SketchSeeker is a highly efficient sketch retrieval system in terms of time and space that obtains excellent similarity results for general input query sketches. With its compression, speed, and accuracy, it has many potential extensions for further application in specific sketch domains.

Acknowledgment

The authors would like to express their appreciation for the assistance and support of all members of the Sketch Recognition Lab at Texas A&M University, especially those who took so much time to provide their help and input. The authors are especially owing to Dr. H.-C. Kum and P. Taele for the useful suggestions they gave.

References

[1] D. Rubine, "Specifying gestures by example," *Comput. Graph.*, vol. 25, pp. 329–337, 1991.

[2] J. O. Wobbrock, A. D. Wilson, and Y. Li, "Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes," in *Proc. 20th Annu. ACM Symp. User Interface Softw. Technol.*, New York, NY, USA: ACM, 2007, pp. 159–168. [Online]. Available: http://doi.acm.org/10.1145/1294211.1294238

[3] T. Y. Ouyang and R. Davis, "ChemInk: A natural real-time recognition system for chemical drawings," in *Proc. 16th Int. Conf. Intell. User Interfaces*, New York, NY, USA: ACM, 2011, pp. 267–276. [Online]. Available: http://doi.acm.org/10.1145/1943403.1943444

[4] T. Hammond and R. Davis, "Ladder, a sketching language for user interface developers," in *ACM SIGGRAPH 2007 Courses*, New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1281500.1281546

[5] O. Atilola *et al.*, "Mechanix: A natural sketch interface tool for teaching truss analysis and free-body diagrams," *Artif. Intell. Eng. Des., Anal. Manuf.*, vol. 28, pp. 169–192, 2014. [Online]. Available: http://journals.cambridge.org/article_S0890060414000079

[6] S. Valentine *et al.*, "Mechanix: A sketch-based tutoring and grading system for free-body diagrams," *AI Mag.*, vol. 34, no. 1, pp. 55–66, 2012.

[7] S. Valentine *et al.*, "Mechanix: A sketch-based tutoring system for statics courses," in *Proc. 24th Innovative Appl. Artif. Intell. Conf.*, 2012, pp. 2253–2260.

[8] M. Eitz, J. Hays, and M. Alexa, "How do humans sketch objects?" *ACM Trans. Graph.—Proc. ACM SIGGRAPH*, vol. 31, no. 4, pp. 44:1–44:10, 2012.

[9] M. Eitz and J. Hays, "Learning to classify human object sketches," in *Proc. ACM SIGGRAPH 2011: Talks*, 2011, Art. no. 30.

[10] M. Eitz, K. Hildebrand, T. Boubekeur, and M. Alexa, "Sketch-based image retrieval: Benchmark and bag-of-features descriptors," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 11, pp. 1624–1636, Nov. 2011.

[11] M. Eitz, K. Hildebrand, T. Boubekeur, and M. Alexa, "An evaluation of descriptors for large-scale image retrieval from sketched feature lines," *Comput. Graph.*, vol. 34, no. 5, pp. 482–498, 2010.

[12] M. Eitz, K. Hildebrand, T. Boubekeur, and M. Alexa, "Sketch-based 3D shape retrieval," in *Proc. ACM SIGGRAPH 2010: Talks*, 2010, Art. no. 5.

[13] R. G. Schneider and T. Tuytelaars, "Sketch classification and classification-driven analysis using fisher vectors," *ACM Trans. Graph.*, vol. 33, no. 6, pp. 174:1–174:9, 2014.

[14] B. Paulson and T. Hammond, "MARQS: Retrieving sketches learned from a single example using a dual-classifier," *J. Multimodal User Interfaces*, vol. 2, no. 1, pp. 3–11, 2008. [Online]. Available: http://dx.doi.org/10.1007/s12193-008-0006-0

[15] M. Eitz, R. Richter, T. Boubekeur, K. Hildebrand, and M. Alexa, "Sketch-based shape retrieval," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 311–3110, 2012.

[16] Z. Sun, C. Wang, L. Zhang, and L. Zhang, "Query-adaptive shape topic mining for hand-drawn sketch recognition," in *Proc. 20th ACM Int. Conf. Multimedia*, ACM, 2012, pp. 519–528. [Online]. Available: http://dblp.uni-trier.de/db/conf/mm/mm2012.html#SunWZZ12

[17] M. Rusiñol and J. Lladós, "Efficient logo retrieval through hashing shape context descriptors," in *Proc. 9th IAPR Int. Workshop Document Anal. Syst.*, New York, NY, USA: ACM, 2010, pp. 215–222. [Online]. Available: http://doi.acm.org/10.1145/1815330.1815358

[18] Z. Sun, C. Wang, L. Zhang, and L. Zhang, "Query-adaptive shape topic mining for hand-drawn sketch recognition," in *Proc. 20th ACM Int. Conf. Multimedia*, New York, NY, USA: ACM, 2012, pp. 519–528. [Online]. Available: http://doi.acm.org/10.1145/2393347.2393421

[19] C. Wang, Z. Li, and L. Zhang, "MindFinder: Image search by interactive sketching and tagging," in *Proc. 19th Int. Conf. World Wide Web*, New York, NY, USA: ACM, 2010, pp. 1309–1312. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772909

[20] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, vol. 2, 1999, pp. 1150–1157.

[21] S. Belongie, J. Malik, and J. Puzicha, "Shape context: A new descriptor for shape matching and object recognition," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 831–837.

[22] M. Oltmans, "Envisioning sketch recognition: A local feature based approach to recognizing informal sketches," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 2007.

[23] H. Ma and D. Doermann, "Adaptive hindi OCR using generalized Hausdorff image comparison," *ACM Trans. Asian Lang. Inf. Process.*, vol. 2, no. 3, pp. 193–218, Sep. 2003. [Online]. Available: http://doi.acm.org/10.1145/979872.979875

[24] T. Di Mascio, M. Francesconi, D. Frigioni, and L. Tarantino, "Tuning a CBIR system for vector images: The interface support," in *Proc. Work. Conf. Adv. Vis. Interfaces*, New York, NY, USA: ACM, 2004, pp. 425–428. [Online]. Available: http://doi.acm.org/10.1145/989863.989942

[25] D. Zhang and G. Lu, "Review of shape representation and description techniques," *Pattern Recognit.*, vol. 37, no. 1, pp. 1–19, 2004.

[26] S. Belongie, G. Mori, and J. Malik, "Matching with shape contexts," in *Statistics and Analysis of Shapes*. New York, NY, USA: Springer-Verlag, 2006, pp. 81–105.

[27] G. V. Pedrosa, S. O. Rezende, and A. J. M. Traina, "Reducing the dimensionality of the sift descriptor and increasing its effectiveness and efficiency in image retrieval via bag-of-features," in *Proc. 18th Brazilian Symp. Multimedia Web*, New York, NY, USA: ACM, 2012, pp. 139–142. [Online]. Available: http://doi.acm.org/10.1145/2382636.2382668

[28] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms," in *Proc. 18th ACM Int. Conf. Multimedia*, ACM, 2010, pp. 1469–1472. [Online]. Available: http://dblp.uni-trier.de/db/conf/mm/mm2010.html#VedaldiF10

[29] L. J. van der Maaten, E. O. Postma, and H. J. van den Herik, "Dimensionality reduction: A comparative review," *J. Mach. Learn. Res.*, vol. 10, no. 1–41, pp. 66–71, 2009.

[30] D. DeMers and G. Cottrell, "Non-linear dimensionality reduction," *Adv. Neural Inf. Process. Syst.*, vol. 5, pp. 580–587, 1993.

[31] L. van der Maaten and G. Hinton, "Visualizing high-dimensional data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, 2008.

Seth Polsley received the B.S. degree in computer engineering from the University of Kansas, Lawrence, KS, USA, in 2014. He is currently working toward the M.S. degree in computer engineering in the Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA.

He is currently a Research Assistant in the Sketch Recognition Lab, Texas A&M University. His research interest in intelligent systems has led to work on multiple sketch-based systems in the domains of education and health.

Jaideep Ray received the B.E. degree in electronics engineering from Jadavpur University, Kolkata, India, in 2011, and the M.S. degree in computer science from the Department of Computer Science and Engineering, Texas A&M University, TX, USA, in 2016.

He is currently with the Sketch Recognition Lab, Texas A&M University, researching approximate hand-drawn sketch matching. His main research interests include scalable systems, applied machine learning techniques, and sketch-based interfaces.

Tracy Hammond received the B.A. degree in mathematics, the B.S. degree in applied mathematics, the M.S. degree in computer science, and the M.S. degree in anthropology from Columbia University, New York, NY, USA, and the Ph.D. degree in computer science with the financial technology option from Massachusetts Institute of Technology, Cambridge, MA, USA.

She is currently the Director of the Sketch Recognition Lab, Texas A&M University, College Station, TX, USA, and a Professor in the Department of Computer Science and Engineering. She is an international leader in sketch and activity recognition, haptics, intelligent fabrics, smart phone development, and computer–human interaction research.