Unsupervised Object Discovery from Images by Mining Local Features Using Hashing

Gibran Fuentes Pineda, Hisashi Koga, and Toshinori Watanabe

Graduate School of Information Systems, The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu-si, Tokyo 182-8585, Japan

Abstract. In this paper, we propose a new methodology for efficiently discovering objects from images without supervision. The basic idea is to search for frequent patterns of closely located features in a set of images and consider a frequent pattern as a meaningful object class. We develop a system for discovering objects from segmented images. This system is implemented by hashing only. We present experimental results to demonstrate the robustness and applicability of our approach.

1 Introduction

Object recognition and discovery from images have been challenging problems in image analysis over the past decades. Typically, objects are represented by either a set of geometric elements such as cones, spheres, and planes (model-based), their contour (shape-based) or their appearance (appearance-based). Then, an object class is modeled by creating an approximate representation (generative models such as Bayesian network [1], and Non-Negative Factorization [2]) or defining an optimal decision boundary (discriminative models, e.g. boosting [3], and SVM [4]) from a set of given examples. In general, these methods scale poorly for very large databases because a) they require some kind of supervision, b) their performance is greatly affected by the high dimensionality of the object representation, and/or c) they are tailored to specific object classes (e.g. faces).

This work attempts to overcome these limitations by efficiently discovering objects from images without supervision. Our assumption is that an object consists of multiple components which are expressed as a set of local image features. To discover object classes without supervision we search for frequent patterns of closely located image features and consider one frequent pattern as a meaningful object class. By searching the known classes, the same approach can be further used for matching a query object with the most similar class, thus enabling the unification of modeling and matching. To simplify the complexity of our approach, we implement it completely by relying on a single technique, namely hashing. We show that hashing can be used to efficiently realize a variety of similarity judgments. Specifically, we demonstrate that the next three kinds of similarity judgments can be implemented by hashing: (1) standard distance-based similarity judgment, (2) distance-based similarity judgment considering the relative size, and (3) matching robust to small variations. By experiment, we prove that our approach can discover diverse object classes robustly against rotation and slide operations as well as small intraclass variations.

This paper is organized as follows. We describe locality-sensitive hashing in Sect. 2. In Sect. 3 we give an overview of our approach. Then, Sect. 4 discusses the detailed description of our system and reports some experimental results. Finally, Sect. 5 concludes the paper.

2 Similarity Judgments by Hashing

Similarity judgment is a fundamental element for pattern recognition in image analysis systems where multiple similarity measures might be necessary because items are defined by many attributes. However, as most image analysis schemes presume only specific spaces and similarity measures (commonly the Euclidean distance), it is not guaranteed that the same scheme will have the same good performance when applied to other spaces and/or similarity measures. The complexity of the system will increase if one decides to support several schemes simultaneously to treat different kinds of similarity measures.

Since hashing techniques have provided an efficient searching mechanism for various similarity judgments that are common in image analysis tasks, we believe that it is possible to construct a simple and efficient image analysis system by using such techniques. Hence, in this paper we consistently rely on hashing techniques inspired by the *locality-sensitive hashing* (LSH) [5].

We describe in detail LSH hereinafter. Let P be a set of points in a d-dimensional space and C be the maximum coordinate value of any point in P. We can transform every $p \in P$ to a Cd-dimensional vector by concatenating unary expressions for every coordinate, that is,

$$f(p) = \mathrm{Unary}(x_1)\mathrm{Unary}(x_2)\cdots\mathrm{Unary}(x_d), \tag{1}$$

where $\mathrm{Unary}(x)$ is a sequence of x ones followed by $C-x$ zeros. A hash function is computed by picking up k bits (which are called *sample bits*) randomly from these Cd bits and concatenating them. This corresponds to partitioning the d-dimensional space into cells of different sizes by k hyperplanes so that near points tend to have the same hash value. As k becomes large, remote points are less likely to take the same hash value because the size of generated cells becomes small. Figure 1 illustrates the space partitioning when $d = 2$, $C = 5$ and $k = 2$. This example presents the hash value of each cell when the second and eighth bits (i.e. x = 2 and y = 3) are selected from the total $2 \times 5 = 10$ bits. By contrast, depending on the result of space division, near points may take different hash values (e.g. point A and point B in Fig. 1). To exclude this failure, multiple l hash functions $h_1, h_2, \cdots h_l$ are prepared in LSH expecting that two points close to each other will take the same hash value at least for one hash function.

Overall, LSH considers that a pair of points with the same hash value are close to each other. We borrow this idea but while this LSH scheme utilizes randomized functions, we define deterministic functions more suitable for our object discovery scheme.

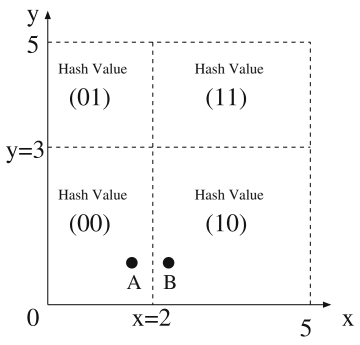

Fig. 1. Space partition by LSH

3 Overview of Our Approach

In this section we present an overview of our approach for discovering objects automatically from a set of images Σ. Each image in Σ consists of several local image features. The underlying idea is to search for frequent patterns of closely located features in Σ and consider each frequent pattern as a meaningful object class. Thereby, our approach runs in four phases described below.

Phase I: By extracting every feature in Σ, we derive a set of object components. We denote this set by $C = \{C_1, C_2, \ldots, C_N\}$.

Phase II: The components in C are classified according to their attributes. A label ID is assigned to each component according to the classification result; the labels are expressed by $\ell_1, \ell_2, \ldots, \ell_M$, where M is the number of component classes.

Phase III: Closely located components are gathered to generate object candidates. Let $T = \{T_1, T_2, \ldots, T_Z\}$ be the set of all object candidates.

Phase IV: We determine object classes by searching frequent patterns in T. A pattern with multiple occurrences is regarded as a meaningful object class. Each object class is represented by the set of component labels that is common in the multiple occurrences.

Figure 2 presents an example of the operation of our approach. First, 11 components from C_1 to C_{11} are extracted from two images. Next, the labels from ℓ_1 to ℓ_6 are assigned to each component; here, similar components have the same label (e.g. both roofs C_3 and C_8 have ℓ_3). Then, the object candidates T_1, T_2, T_3 and T_4 are generated by gathering closely located components. Finally, the Class 1 ("tree") and Class 2 ("house") are regarded as meaningful object classes because each of them has two occurrences in the images; "tree" is represented by ℓ_1 and ℓ_2 whereas "house" is represented by ℓ_3, ℓ_4 and ℓ_5.

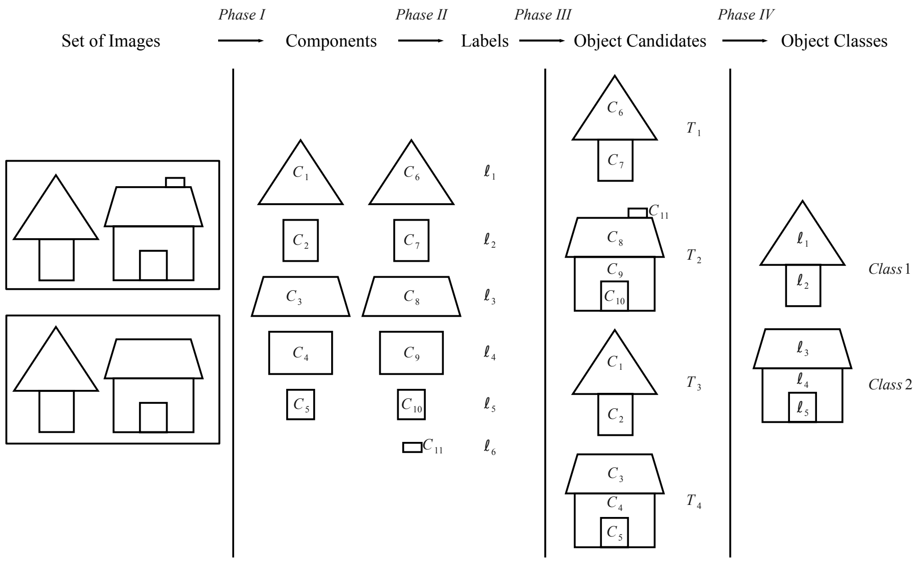

Fig. 2. Intuitive example of our object discovery approach

4 Object Discovery from Segmented Images

This section describes the implementation details of our system. As inputs, our system receives images preprocessed by segmentation and color quantization algorithms; each region of the preprocessed images is regarded as a single local feature. In addition, we assume that an object consists of closely located regions and does not overlap with other objects. Thus, our system performs three kinds of similarity judgments: (1) distance-based similarity judgment considering the relative size in Phase II, (2) ordinary distance-based similarity judgment in Phase III, and (3) matching robust to small variations in Phase IV. We modify the LSH scheme described in Sect. 2 to implement the first two kinds of similarity judgments whereas for the third kind we extend the standard hashing for exact matching to perform matching robust to small variations.

4.1 Phase I: Extraction of Components

In order to extract object components, we first identify regions in Σ that correspond to background. Then, we consider as object components all regions in Σ that are not identified as background . Let us denote this set of components by C. Although the discrimination between background and foreground regions is difficult and sometimes requires supervision, when the background is non-textured or represents the largest regions of the image, the automatic identification of the background becomes possible.

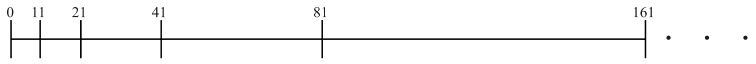

Fig. 3. Location of sample bits on the real line when $\alpha = 10$, $\beta = 2$ and $i = 1$

4.2 Phase II: Labeling of Components

The components in C are labeled according to their color and size such that components of the same color with similar size are assigned the same label. For this purpose, since color quantization is performed in the preprocessing phase, we may cluster components of different color separately. Therefore, the components of the same color are hashed according to their sizes. However, the similarity between sizes should be relative to their absolute value. Hence, the hash functions are defined by selecting sample bits at intervals proportional to the distance from the origin. That is, for the i-th hash function $(1 \leq i \leq l)$, the sample bits are set as follows.

$$\begin{array}{c}\text{Location of sample bits}\\ h_i : \alpha + i,\, \alpha\beta + i,\, \alpha\beta^2 + i,\, \ldots\, , \alpha\beta^k + i,\end{array} \tag{2}$$

where α determines the position of the first sample bit and β is the growth factor of the intervals ($\alpha > 0$ and $\beta > 1$). Figure 3 illustrates the location of the sample bits on the real line when $\alpha = 10$, $\beta = 2$ and $i = 1$. Note that the intervals between the sample bits become wider as they become farther from 0.

To cluster similar components we apply the CENTER algorithm [6]. CENTER makes graphs where vertices are components and an edge is made between a pair of components if they have the same hash value at least for one hash function. Then, graphs are partitioned in such a way that in each cluster the center node has an edge to the other nodes. This process is carried out by following the next steps.

Step I: For each color, pick up the biggest unchecked component B in C.
Step II: Select all the unchecked components that have an edge to B and merge them into the same cluster.
Step III: Mark all the merged components as checked.
Step IV: Repeat step 1-3 until all the components have been checked.

After this process, we assign the labels $\ell_1, \ell_2, \ldots, \ell_M$ to the clusters according to the size of the center components such that ℓ_1 and ℓ_M corresponds to the largest and smallest component respectively.

4.3 Phase III: Generation of Object Candidates

We generate object candidates by clustering closely located components. The nearness between two separate components is determined by the Euclidean distance among their pixels. Therefore, we hash all pixels in every component in C. The hash functions are defined by selecting sample bits at equal intervals of a parameter I. Consequently, the number of sample bits k is expressed as follows.

$$k = \frac{X_{max} + Y_{max}}{I}, \tag{3}$$

where X_{max} and Y_{max} denote respectively the number of columns and rows of the given image. We define the I hash functions so that the sample bits do not coincide one another at all in the following way.

$$\begin{array}{l} \text{Location of sample bits} \\ h_1 : 1,\, I+1,\, 2I+1,\, \cdots\, (k-1)I+1 \\ h_2 : 2,\, I+2,\, 2I+2,\, \cdots\, (k-1)I+2 \\ \quad\vdots \\ h_I : I,\quad 2I,\quad 3I,\quad \cdots\quad kI \end{array}. \tag{4}$$

For generating object candidates, we adopt the next rule.

Rule 1. *Two separate components C_i and C_j $(i, j = 1, \ldots, N)$ are clustered into the same object candidate if one pixel in C_i and one pixel in C_j have the same hash value at least for one hash function.*

Each object candidate T_i $(1 \leq i \leq Z)$ is represented by a vector

$$T_i = [\upsilon_1, \ldots, \upsilon_M], \tag{5}$$

where υ_r $(1 \leq r \leq M)$ denotes the number of components with label ℓ_r in the object candidate T_i. For example, the object candidate T_1 in Fig. 2 is generated from the components C_6 and C_7 (with labels ℓ_1 and ℓ_2 respectively) because they are close to one another. In this case, the representation of the object candidate becomes $T_1 = (1, 1, 0, 0, 0, 0)$.

4.4 Phase IV: Determination of Object Classes

In order to determine meaningful object classes, we search for multiple occurrences of similar object candidates. We judge object candidates as similar if their primary components are the same. Standard hashing is applied to accelerate this process. To compute the hash value for an object candidate T_i $(1 \leq i \leq Z)$, we first concatenate the elements υ_r $(1 \leq r \leq M)$ of T_i, that is,

$$cat(T_i) = \upsilon_1 \upsilon_2 \cdots \upsilon_M, \tag{6}$$

where $\upsilon_1, \upsilon_2, \cdots, \upsilon_M$ are expressed by λ bits so that $|cat(T_i)| = \lambda M$. In order to avoid small intra-class variations, we generate J hash values for T_i by ignoring the $\xi, \xi+1, \ldots, \xi+J-1$ smallest components from $cat(T_i)$, where ξ presents the maximum integer such that the sum of the size of the ξ smallest components in T_i does not exceed the μ% of the whole size of T_i. After computing the J hash values for each object candidate, we obey the next rule to determine meaningful object classes.

Rule 2. *Two object candidates are classified into the same cluster if at least one of their J hash values is the same.*

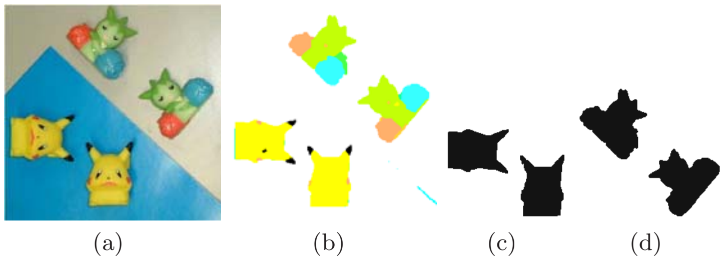

Fig. 4. Discovery of objects with rotation and slide variations: (a) original image, (b) preprocessed image, (c) class 1 and (b) class 2

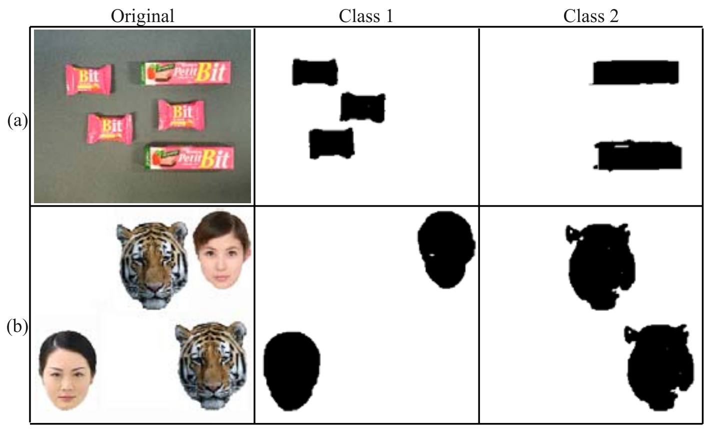

Fig. 5. Discovery of objects with intraclass variations

After clustering the object candidates, we regard as meaningful object classes only those clusters with multiple object candidates. We represent each of these classes in the same form as (5), where υ_r $(r = 1, \ldots, M)$ stands for the number of components with label ℓ_r that are common to all object candidates of the given class. For instance, T_2 and T_4 in Fig. 2 are classified into the same cluster by ignoring C_{11}, which is extremely small relative to the size of T_2. Then, since this cluster has two object candidates, it is regarded as a meaningful object class (Class 1) and represented by ℓ_3, ℓ_4 and ℓ_5, i.e., $\upsilon = (0, 0, 1, 1, 1, 0)$. Note that ℓ_6 is not included, because it is not a component of T_4.

4.5 Experimental Results

For the experiments, each image was segmented by using the MST-based algorithm [7] and then a color quantization was performed. Finally, we considered the extremely big regions of the image as background and remove them so that objects were isolated. An example of the segmentation, quantization and background removal can be seen in Fig. 4(a) and 4(b).

Initially, we evaluated the robustness of our system against rotation and slide operations. To that end, we applied our system to an image that contains two instances of two different object classes (Fig. 4(b)). Note that the orientations of the two instances of the same class differ approximately by 90 degrees. Since our system does not consider the exact location relation between components, both object classes (Fig. 4(c) and 4(d)) were successfully discovered despite these transformations.

We also evaluated the robustness of our system against intraclass variations. In Fig. 5 we present two examples of this evaluation. The first example consists of an image containing two kinds of candy (Fig. 5(a)). The other example consists of two kinds of faces: human faces and tiger faces (see Fig. 5(b)). Note that the two human faces are different. In both examples our system derived two object classes successfully. The columns Class 1 and Class 2 in Fig. 5 illustrate the instances of each derived class in each image. As we can observe our system can cope with small intraclass variations such as faces of different subjects.

5 Conclusions

We proposed a new methodology for discovering object classes from images which can discover and recognize diverse object classes without supervision. This methodology can be suitable for indexing and searching objects in large image databases with diverse contents. We demonstrated that frequent patterns of local image features can lead to discover meaningful object classes. Our approach can be completely implemented by only hashing which simplifies its implementation and at the same time enables the integration of various similarity measures. We proved by experiment that our approach is robust against rotation and slide operations as well as small intraclass variations.

References

1. Bar-Hillel, A., Weinshall, D.: Efficient learning of relational object class models. International Journal of Computer Vision 77(1–3), 175–198 (2008)
2. Lee, D.D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature 401, 788–791 (1999)
3. Opelt, A., Pinz, A., Fussenegger, M., Auer, P.: Generic object recognition with boosting. IEEE Transactions on Pattern Analysis and Machine Intelligence 28(3), 416–431 (2006)
4. Heisele, B., Serre, T., Poggio, T.: A component-based framework for face detection and identification. International Journal of Computer Vision 74(2), 167–181 (2007)
5. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: Thirtieth Annual ACM Symposium on the Theory of Computing, pp. 604–613 (1998)
6. Haveliwala, T.H., Gionis, A., Indyk, P.: Scalable techniques for clustering the web. In: Third International Workshop on the Web and Databases, pp. 129–134 (2000)
7. Tsunoda, N., Watanabe, T., Sugawara, K.: Image segmentation by adaptive thresholding of minimum spanning trees. Transactions of the Institute of Electronics, Information and Communication Engineers J87-D-II(2), 586–594 (2004)