Icon Scanning: Towards Next Generation QR Codes

Itamar Friedman
Technion
Haifa, Israel
itamarf@technion.ac.il

Lihi Zelnik-Manor
Technion
Haifa, Israel
lihi@ee.technion.ac.il

Abstract

Undoubtedly, a key feature in the popularity of smart-mobile devices is the numerous applications one can install. Frequently, we learn about an application we desire by seeing it on a review site, someone else's device, or a magazine. A user-friendly way to obtain this particular application could be by taking a snapshot of its corresponding icon and being directed automatically to its download link. Such a solution exists today for QR codes, which can be thought of as icons with a binary pattern. In this paper we extend this to App-icons and propose a complete system for automatic icon-scanning: it first detects the icon in a snapshot and then recognizes it.

Icon scanning is a highly challenging problem due to the large variety of icons (∼500K in App-Store) and background wallpapers. In addition, our system should further deal with the challenges introduced by taking pictures of a screen. Nevertheless, the novel solution proposed in this paper provides high detection and recognition rates. We test our complete icon-scanning system on icon snapshots taken by independent users, and search them within the entire set of icons in App-Store. Our success rates are high and improve significantly on other methods.

1. Introduction

In recent years, the concept of information transfer via visual search is becoming progressively more common. For example, in SnapTell [5] the user snaps a picture of the cover of a book or DVD and the system automatically recognizes the relevant product and delivers its related information. Google-Goggles [3] recognizes text, landmarks and certain objects. QR codes [1] are binary patterns that one snaps in order to be directed to a relevant link. As seen in Figure 1, QR codes are not visually appealing. Nevertheless, the increasing popularity of QR codes suggests this form of visual search is being endorsed by the crowds. Hence, we wish to adopt this idea and extend it to general icons. In particular, our solution focuses on iOS apps' icons.

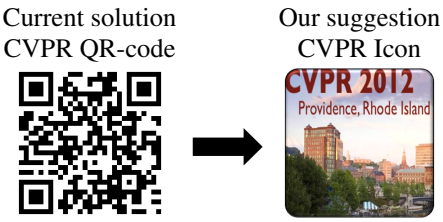


Figure 1. **From QR-codes to icons.** Nowadays to quickly get to the CVPR web-page one can scan its corresponding QR code. Instead, we propose scanning the conference icon.

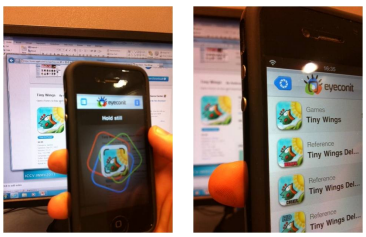


Figure 2. **App-scanning.** In our system downloading an application amounts to hovering over the corresponding icon with the mobile device (left). Our system automatically detects and recognizes it (right).

In this paper we describe an "App-scanner". As illustrated in Figure 2, to download an application to one's mobile device all that needs to be done is to hover with the device's camera in-front of the corresponding icon, or to take a snapshot of it (same methodology as for QR codes). The system then automatically detects the icon in the snapshot, and recognizes it within the entire App-Store icon database.

A first attempt in this direction has been proposed by AppSnap [2] – an iPhone application that lets the user install any application by taking a picture of its icon. AppSnap does not address detection. Instead, the user is requested to manually mark the outline frame of each snapped icon. A test we held showed that when given accurately marked icons, AppSnap's icon recognition rates are high, which we find very encouraging. However, we further noticed that when the accuracy of the icon's snapshot is reduced, e.g,. when 5% of the snapped icon is missing or replaced with

part of the background wallpaper, the recognition rates drop drastically to almost null. Therefore, AppSnap recognition system is not robust enough to be part of a complete scanning system. Applications such as Google-Goggles [3] and TinEye [6] do not address icon detection and therefore are confused by the pattern of the background wallpaper. Furthermore, they are not designed to handle the noise introduced when taking pictures of a screen.

The first contribution of this paper is a novel algorithm for automatic detection of icons in a snapshot (Section 2). This removes the need for manually marking the icon's boundary, thus, facilitating significantly the user experience. To handle the large variety of icon and background patterns we first apply multiple different boundary detectors, and then aggregate and prune them to obtain the final result. Our experiments show this leads to high detection rates in real images taken by multiple different users.

The second contribution of this paper is a recognition system for icons that originate from photos of screens (Section 3). We analyze the visual properties unique to App-icons and propose a descriptor, that outperforms conventional descriptors such as SIFT [16, 7]. We further explore (in Section 3.2) the challenges introduced when handling pictures of screens and suggest a solution that is robust to them. Our experiments show this leads to recognition rates significantly higher than those of AppSnap.

Finally, the third contribution of this paper is presenting a complete system for icon-scanning. We have implemented the system as an App for iPhone and iPad, installed it on several devices, and let independent users play with it, thus collecting data for testing. In Section 4 we discuss implementation details. Experimental validation is presented throughout the paper.

2. Icon detection

Detecting icons within a screen image is a challenging variant of image segmentation and boundary detection. The difficulty stems from the variability in patterns and the low quality of screen images taken by a hand-held device. Furthermore, as is customary with QR codes, we want to detect the icon while the camera hovers over it, i.e., the user need not take a snapshot. Instead, our system should search for an icon continuously and take a snapshot when one is detected. This implies icon detection should run in real-time. Accordingly, the solution proposed in this section presents both high success rates as well as real-time performance.

By manually analyzing multiple screen images we have noticed that often the icon is easily separable from the background in one color channel, while in the other channels the boundaries are less vivid. We cannot, however, tell in advance which are the useful channels. Therefore, inspired by [17], we first generate multiple boundary maps by analyzing different color channels via several edge detectors. Next, the detected edges are merged into contours which serve as candidates for the icon boundary. Finally, we rely on the unique structural properties of iOS's App-icons to determine the final detection of the icon's boundary. We next describe each step in detail.

Edge detection: Our initial step is edge detection. Recalling our detection should run in real-time we cannot use any sophisticated edge detectors such as [8]. Instead, we adopt the time-efficient Canny edge detector [9]. We denote by $C^{1,\ldots,9}$ the RGB, HSV and Lab color channels of the input image. We compute the Canny edge response of each of the 9 color channels C^i. Next, we produce 50 edge images $E^i_{1,\ldots,50}$ for each channel C^i by thresholding the response image at $[5, 10, \ldots, 250]$. We use image opening and closing operations to reduce noise and connect between nearby edges. As shown in Figure 3, each edge image E^i_j reveals different boundaries. Hence, we further take $m = 50$ pseudo-random linear combinations of the edge images E^i_j, yielding our final edge images $E^+_{1,\ldots,m}$.

Finding potential contours: The next step is aggregating the detected edge pixels into contours $PC_{1,\ldots,P}$, that serve as candidates for the icon boundary. This is performed in two stages, as shown in Figure 4. First, we join edge pixels into contours, and exclude over-fragmented contours. Then, recalling that App-icons have a characteristic shape, we compute the convex hull of each contour and exclude candidates whose shape contradicts these characteristics.

We connect edge pixels (presented in Figure 4.(b)) into contours using Suzuki's contour detector [19]. As shown in Figure 4.(c), this step yields multiple contours for each edge

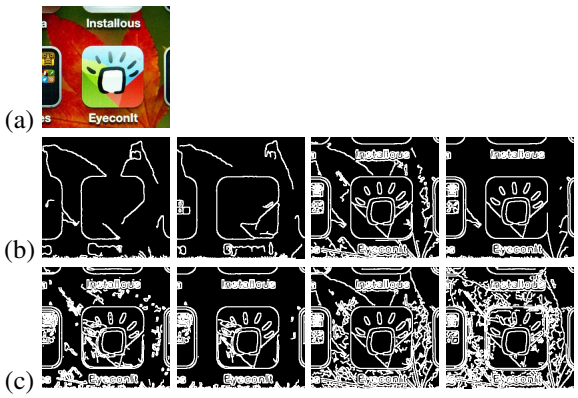
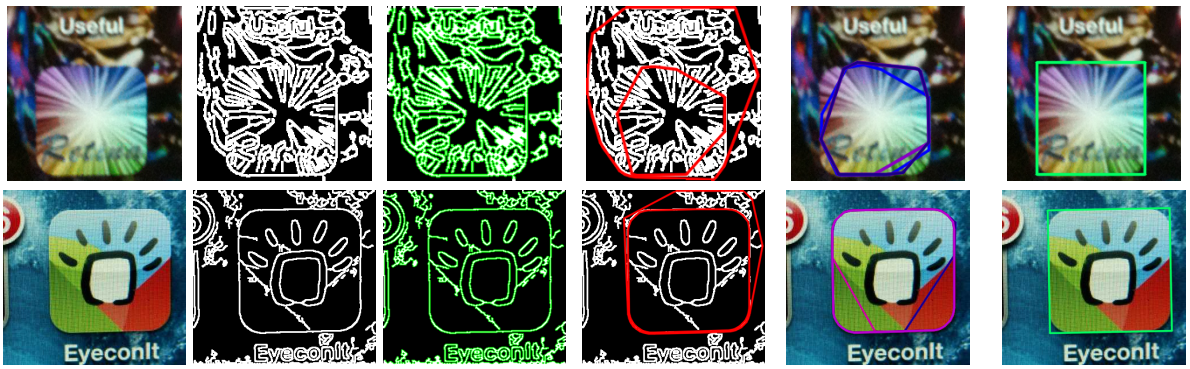


Figure 3. **Edge images.** (top) Input image. (middle) Examples of edge responses E^i_j obtained for different color channels. While some boundary pixels are detected in one response they could be missed in another. Hence, no single edge response suffices for boundary detection. (bottom) Examples of combined edge responses $E^+_{1,\ldots,m}$. By taking linear combinations of the initial edge responses we wish to obtain edge images that include all boundary pixels (as well as other edges).



Input (a) Edges (b) Initial contours (c) Convex-hull fitting (d) Potential contours (e) Final detection result (f)

Figure 4. **Finding candidate contours.** (a) Input images. (b) Examples of our combined edge images E^+. Many edge pixels do not belong to the icon boundary. (c) The edges are merged into contours. (d) A convex hull is fitted to each contour to close gaps. Over-fragmented and concave shapes are excluded. (e) The final candidate boundary contours $PC_{1..P}$, obtained by excluding contours whose shape does not match that of an icon. It can be seen that only closed polygons with a small number of segments are kept. Most of the potential contours fit the icon's boundary very well. (f) our grading method chooses the best candidate bounding-box.

image, of all shapes and forms. To exclude overly fragmented contours we fit each contour with a polygon [11] and filter out contours whose corresponding polygon has over 64 line segments (this number was tuned empirically). This process rejects more than 40% of the contours.

Finally, we adopt the following assumptions: (1) Icons have convex structures. (2) May have non-continuous edges. (3) Should have a small number of ribs. Therefore, we compute for each contour a convex hull approximation [18], thus closing non-continuous edges. Following assumptions (1) and (2) we reject concave contours. Example results are shown in Figure 4(d). Furthermore, To conform with assumption (3) we simplify each convex-hull by fitting to it a polygon [11] with no-less than 4 line segments. This yields the colored potential contours $PC_{1,\ldots,P}$ displayed in Figure 4.(e).

Selecting the best contour: To choose the best candidate we note that icons can appear as squares, or as mild affine transformations of squares. Therefore, for each candidate contour PC_p we find the tightest rectangular bounding box BB_p. We then evaluate its squareness by computing:

$$\begin{aligned} AreaRatio_p &= \frac{Area(PC_p)}{Area(BB_p)} \\ RibsRatio_p &= 2 \cdot \left| \frac{width(BB_p) - height(BB_p)}{width(BB_p) + height(BB_p)} \right|. \end{aligned}$$

We grade each candidate as follows:

$$Grade_p = AreaRatio_p - \lambda \cdot RibsRatio_p,$$

where $\lambda = 0.4$. The bounding box with the highest grade is selected as the final bounding box, see Figure 4.(f).

Recall, that our detection should run continuously in real-time, while the user hovers with the camera over the icon. Therefore, not all input images actually include an icon. Our system should reject such images. Empirical testing revealed that the detection is trustworthy when $0.92 < \text{RibsRatio} < 1.08$ and $\text{AreaRatio} > 0.7$. We hence exclude images whose best bounding box does not comply with these values, and declare a detection otherwise. When no bounding box passes this test we repeat the entire process using a new set of edge images E^+. We allow 4 attempts of the said above and ask for user intervention if all failed.

Empirical evaluation of icon detection To evaluate our approach we have collected a database of 900 snapshots, taken by different users and from different screens, while using our application. 500 of the snapshots include an icon (of 228 different apps) and 400 do not. We then manually created ground-truth masks for each snapshot. This database will be made public [1].

Running on Intel(R) Xeon(R) CPU E5410 @ 2.33GHz, 1.7G RAM, while testing 128×128 images we show a fast detection run time of $148ms \pm 42ms$ when an icon exists in the image and $259ms \pm 3ms$ when no icon exists. The respectively large variance, and the difference between the two cases derives from the auxiliary mechanism that automatically enlarges the set E^+ if no icon was found.

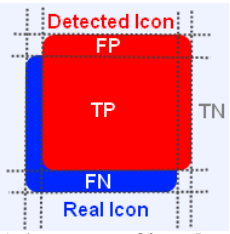
Table 1 summarizes quantitatively the results obtained by our system. As can be seen, the detection rates are extremely high. Most importantly, we never declare an icon detected when there isn't one in the image. This is an important behavior as we do not wish the system to take a snapshot and head on to recognition unless an icon had correctly been detected. Missing an icon when it exists

[1] "http://cgm.technion.ac.il/"

True Positive	94.2%
True Negative	100%
False Positive	0%
False Negative	5.8%

Table 1. **Icon detection results.** Our system makes no false positive detections while it detects most of the icons.

(false-negative) is less of a problem. In such cases we resort to asking the user to either try again or to mark the icon manually. We further measured the accuracy of the detected boundary using the scheme displayed in Figure 5. The boundary detection accuracy is high, providing a good starting point for recognition.



True Positive	94.58%
True Negative	94.57%
False Positive	5.42%
False Negative	5.43%

Figure 5. **Accuracy of icon boundary detection.** When an icon is detected we measure the accuracy of the boundary by comparing the outline of detected icon with the ground-truth mask. As can be seen, our system detects the icons with high precision.

Figure 6 visualizes some successful detection results. Even though the images are of low quality, as screen snapshots often are, and in spite of the challenging background patterns, our algorithm successfully detects the icons.

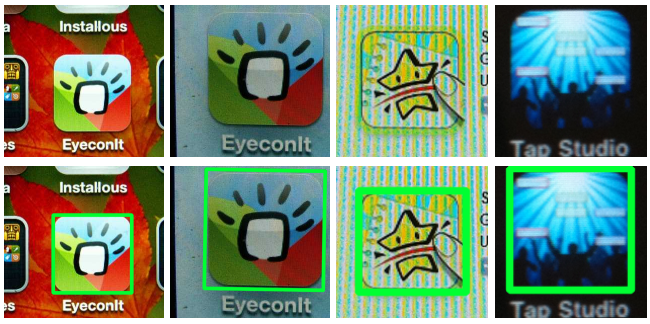


Figure 6. **Detection results.** (top) Input images suffer from severe degradation due to the screen snapshot process. (bottom) Our detection results - our algorithm successfully detects the icons in spite of the challenging input.

3. Icon recognition

Given the detection result our next goal it to find the corresponding icon in the App-Store database. This problem is somewhat related to finding similar images [14, 15] or near-duplicates [10] where visually resembling images are sought. However, icon recognition is different. As shown in Figure 7, often totally different Apps have highly similar icons. While finding similar-images and near-duplicates aim at robustness to small variations, icon recognition requires capturing the differences. The intuitive solution of correlation is unsatisfactory since it requires correlating each query with the multitude of icons in the database, i.e., overly long run-time which is not acceptable for real applications. We hence seek a more compact representation for icons that will lead to efficient recognition.

TangoVideo FindEurope TouchWords

Figure 7. **Similarity between icons of different Apps.** Often icons of different Apps vary only slightly. Methods for finding similar images aim at being robust to such gentle variations and hence are not a good fit for the problem. Icon recognition requires capturing these subtle differences.

To evaluate the quality of each suggested representation we have collected a set of 250 correctly detected icons (of 42 different apps) and search them within the entire set of $\sim 500K$ icons in App-Store.

3.1. A descriptor for icons

Capturing appearance: Our goal is to find a compact image descriptor that captures subtle differences between icons. This implies one should avoid representations based on a sparse set of features, such as Bag-of-Words [13]. Instead we wish to define a descriptor based on the entire icon data. One of the most successful descriptors of this type is SIFT [16] and its color-based version Color-SIFT [12]. These descriptors are known to show minor performance reduction due to small changes in the position of the interest point. To test its usefulness for icon recognition we treat the icon as a single feature and compute a single SIFT descriptor over the entire icon region. As reported in Table 2 the resulting recognition rates are mediocre.

Analyzing the causes of failures we have made two observations, which led to two modifications of the SIFT descriptor. First, the SIFT descriptor includes a Gaussian weighting which assigns low importance to pixels far from the icon center. This is not well suited for icons, which occasionally differ only in the boundaries. For example, as shown in Figure 8 the Cassina icon differs from the Imasys icon in the white text near its top border. Similarly, Shazam for iPad differs from Shazam for iPhone in the surrounding gray outline. Hence, the first modification we apply is to remove the Gaussian weighting step in SIFT.

Second, when dealing with icons one should exclude the corners of the image. This is since the corners of the icons are rounded, thus the image includes background pixels near them. Furthermore, as illustrated in the right panel

Figure 8. **Icon boundaries.** (left) Often icons differ only in regions close to their boundary. A good descriptor should hence consider the entire icon without reducing the importance of boundary pixels, as is done in SIFT. (right) Frequently, various visual symbols are attached to one of the icon's corner, hence, the corner regions should be excluded from the icon descriptor.

Descriptor	Length	Correct best match	Correct match within top 50
AppSnap		64.4%	69.6%
SIFT	128	68.8%	82.8%
Color-SIFT	384	69.6%	84%
Modified-SIFT	96	76%	86%
Color-Modified-SIFT	168	**77.2%**	**88.4%**

Table 2. **Recognition results for various descriptors.** Top 3 rows show recognition results of previous approaches. The bottom rows show results obtained with the descriptor proposed here. As can be seen, our modification to the SIFT descriptor leads to a significant improvement in finding the correct match.

of Figure 8 often visual symbols are attached to one of the icon corners, e.g., to mark the number of messages waiting within the App. Therefore, the second modification we apply to SIFT is excluding the corner regions. Rather than using all the bins of the standard 16×16 SIFT grid, we use only 12 bins, excluding the corner bins. This implies our Modified-SIFT descriptor is a vector of length 96 rather than 128. As shown in Table 2, testing with this Modified-SIFT descriptor improved recognition rates significantly in comparison to the standard SIFT.

Capturing color: So far our descriptor was based on image gradients only and hence does not capture variations in color. As shown in Figure 9, color is often an important cue for differentiating between icons. To incorporate color information we could adopt the Color-SIFT [12] approach and compute our descriptor on three color channels obtaining a $96 \times 3 = 288$ long vector. Furthermore, Color-SIFT is based only on gradients, hence, it fails to distinguish between icons with very low frequencies, such as a flat green icon and a flat red icon. We next propose a more compact representation to capture color, which tackles this problem.

We represent the icons in the Lab color-space, centering

Shazam Encore Shazam Red

Figure 9. **Color differences.** Some icons have a very similar shape, but differ in color. A good descriptor must therefore consider also color information.

and normalizing each channel around 0 (i.e., each channel is spanned between -1 and 1). Even good quality images taken of screens often suffer from color obliquity. We have noticed, however, that typically the deformation in colors is exhibited mostly in the low values of each of the Lab channels. That is, when two icons differ in the close-to-$|1|$ values of the Lab channels they are more likely to be different, than when they differ in the close-to-0 values. To give more importance to the extreme color values we compute for each region the absolute mean positive value and absolute mean negative value for each channel. This yields 6 values per region. When comparing descriptors, taking the $L2$ distance between the absolute mean values emphasizes the extreme values more than the low values. Our final icon descriptor, named Color-Modified-SIFT is therefore of length $96 + 6 \times 12 = 168$. It further improves recognition rates, as shown in Table 2.

3.2. Dealing with screen images

The query icons are typically captured by taking snapshots of a screen. As illustrated in Figure 10 the snapped icons are of deteriorated quality, more than a slight obliquity. Some regions are over-saturated and blur is evident due to the unstable hand motion when taking a picture with a hand-held device. These deformations could interfere with recognition. For example, out of the six screen snapshots in Figure 10 both TinEye [6] and Google-Goggles [3] successfully recognize only two.

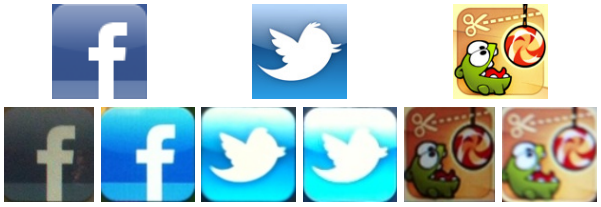
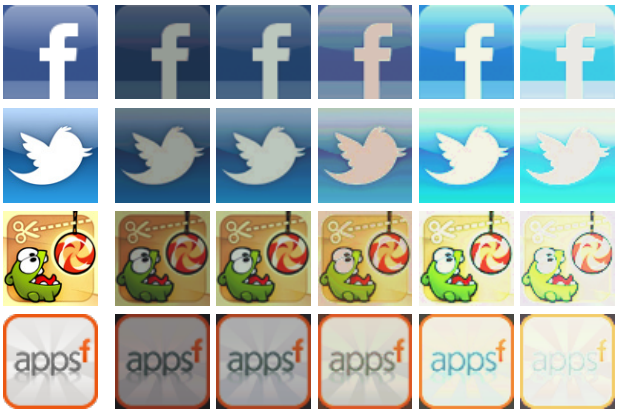


Figure 10. **Noisy input**. (top) Three original icons. (bottom) Examples of screen snapshots of these icons. Since the query icons are obtained by taking snapshots of a screen their colors are often wrong, they could be saturated and/or blurred.

To overcome these phenomenons one could try to deblur the query [22] and correct its color by removing saturated regions [21]. We have found that common algorithms do not always perform well for our purposes since often ringing effects are introduced in the process, resulting in wrong

descriptors. Instead we adopt the opposite approach. Rather than "cleaning" the query we add artifacts to the database.

For each original icon in the App-Store database we generate five deformed versions. These instances are obtained by adding different levels of blur and by modifying the image color curve to be more or less saturated. Examples of the deformations we introduce are presented in Figure 11. Our final database consists of those five deformed versions of each icon. The color deformations we utilize are learned from examples, as described next.



Original Color-deformed icons added to our DB

Figure 11. **Extended database**. To enable recognition of noisy input images we add to the database color-deformed versions of the original icons.

Learning deformation models. To learn the best set of deformations we have collected a set of 250 real screen snapshots of icons, taken by our users. For each snapshot we have found the corresponding original icon, resulting in a training set of 250 image pairs. Our system considers two types of deformations: blur and tone-mapping transformations. The best K deformation models are then learned via a two stage approach. The first stage searches greedily the space of transformations for a good representative set. To guarantee reasonable run-time, we consider at this stage only transformations with a small number of parameters. The second stage refines the initial models using an EM-like approach.

Greedy initialization. We model blur as smoothing the image with a Gaussian kernel with standard deviation σ. Color transformation is modeled as multiplying each color channel c by a constant α^c. For each original icon we generate multiple deformed versions by combining blur and color transformation. In our implementation we use three blur kernels with $\sigma = 0.65, 0.95, 1.25$ and 14 values for each color multiplier α^c in the range $[0.65, 1.3]$. We consider only transformations where $|\alpha_i - \alpha_j| < 0.25$, hence, the total number of deformations we generate for each original icon is 3612.

Denote by I_o an original icon and by I_s the corresponding screen snapshot. Let $X_i, i \in [1, 3612]$ be the generated deformed versions of the icon with deformation parameters $T_i = [\alpha_i^1, \alpha_i^2, \alpha_i^3, \sigma_i]$. To find the best representative set of deformations we compute the similarity between each noisy snapshot I_s to all the corresponding deformed icons X_i:

$$sim(I_s, X_i) = exp\left(-\frac{\|I_s - X_i\|^2 - \tau}{\tau}\right)$$

where $\tau = min_{i,j}\|I_s(j) - X_i(j)\|^2$ is the minimum distance over all the icon pairs j in the training set, over all color deformations T_i. We then compute for each deformation T_i the total similarity over all icon pairs j:

$$totalsim(T_i) = \sum_j sim(I_s(j), X_i(j))$$

A good representative set of transformations T_i is obtained by finding all local maxima points of $totalsim$ over the parameter space T_i. Note, that the number M of local maxima points is data dependent. To find the best combination of K transformations we search greedily over all possible sets of K transformations, and select the combination with maximal $totalsim$. Our experiments revealed that $K = 8$ is sufficient (adding more transformations resulted in negligible increase in $totalsim$). However, due to run-time and memory limitations, in our implementation we use $K = 5$.

K-models clustering. The transformations obtained in the previous step were selected among a discrete set of limited color deformations. To further refine the set of transformations we consider, we next propose an EM like approach for learning the best set of K models. First, for each original icon in our training set we generate K blurred versions using the gaussian kernels obtained in the previous stage. We extend our training set to include these blurred versions resulting in 250 groups of $K + 1$ images for training, i.e., each group consists of a screen snapshot and K blurred version of the original icon.

Next, to allow for more elaborate color deformations than those of the previous stage, we consider all possible tone-mapping functions. This is done as follows. All our images are stacked column wise while concatenating the three RGB color channels into a single column vector. For each color channel c we construct a matrix representation M^c of size $N^2 \times 256$, where N is the number of pixels. We set $M^c(p, q) = 1$ if pixel p has intensity q in color channel c, and 0 otherwise. Using this matrix representation, all possible tone mappings can be obtained by taking the product between M^c and a vector β^c:

$$I = \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} M^R & 0 & 0 \\ 0 & M^G & 0 \\ 0 & 0 & M^B \end{bmatrix} \begin{bmatrix} \beta^R \\ \beta^G \\ \beta^B \end{bmatrix} = M\beta \quad (1)$$

Figure 12. **Icon scanning**. Examples of queries and the retrieved results. Even though the input is often saturated or blurred the system successfully detects the icons, and others similar to it. More results including failures can be found in the supplemental.

In this representation, when the left-hand-side vector I and the matrix M are constructed of the same image, there exists a solution for β which guarantees equality. When I and M are constructed from different images, we can find the best solution for β in the least-squares sense and compute the representation error as:

$$Error = \|I - M\beta\|^2. \tag{2}$$

We use this representation to model the transformation between each snapshot image I_s and the corresponding original icon I_o, by solving for β such that $I_s = M(I_o)\beta$. Hence, finding a representative set of K deformations reduces to finding a set of K vectors β_k. This is done via an EM like framework.

Let r_{jk} be an indicator function equal to 1 when image pair j is associated with model k. We wish to minimize the following objective:

$$J = \sum_{j=1}^{N}\sum_{k=1}^{K} r_{jk}\,\|I_s(j) - M(I_o(j))\beta_k\|^2 \tag{3}$$

where N is the number of image groups in our training set.

We initialize the vectors β_k according to the K transformations computed in the previous stage. Then, in the Expectation phase, each pair of images from our training set is associated with the vector β_k which yields the minimal $Error$ of Eq. (2):

$$r_{jk} = \begin{cases} 1 & \text{if } (k = \arg\min_l Error(\beta_l)) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

In the Maximization phase we can separate the problem into K optimization problems:

$$J_k = \sum_j r_{jk} \cdot \|I_s(j) - M(I_o(j))\beta_k\|^2 . \tag{5}$$

Correct best match within	top 1	top 50
Database of original icons only	44.4%	50.8%
Our extended database	77.2%	88.4%

Table 3. Recognition results with and without our extension to the database, using our Color-Modified-SIFT descriptor. This shows our extension to the database is crucial for the system to function successfully.

Setting the derivative of J_k to zero with respect to β_k, yields a set of linear equations that can be solved in closed form.

The proposed EM framework is guaranteed to converge to a local minimum. We initialize it wisely using the transformations learned in the previous stage. Our empirical evaluation indicates that we obtain a good minima point.

Empirical evaluation of icon recognition The recognition rates presented in Table 2 were all obtained using the extended database. When removing the deformed icons from the database and testing with only the original icons, the success rates drop dramatically for all descriptors. See Table 3. We hence conclude that using the extended database is crucial for obtaining acceptable recognition rates.

4. Implementing a complete system

The detection and recognition algorithms have been implemented in C++ and use openCV2.2 [20]. Our system has been successfully installed and tested on various versions of iPhone and iPad.

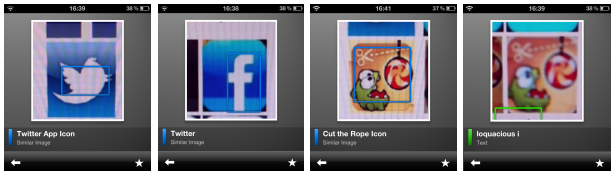
The detection module requires 330ms on iPhone4 to process an image. Once an icon has been detected its descriptor is computed (which takes 90ms) and sent to a server

for matching against the database. Matching is performed using the MySQL tree [4] constructed a-priori for the entire database. The entire recognition process takes ~ 4 seconds.

Figure 12 presents a few of our results (more examples are in the supplemental). Note, that our system finds the correct match even-though the input image is of low quality. The retrieved icons are often highly similar visually, which suggests our descriptor captures appearance nicely.

Finally, we further tried to evaluate the icon-scanning performance of existing systems for visual retrieval. TinEye is a web-app that allows users to search for similar images to a selected input image. When provided with an image of an original icon, TinEye usually successfully finds replicas of its image on the web. However, when tested on real input from our database, i.e., the queries were snapped of screens by real users and are hence noisy, TinEye fails on almost all of the examples we're tried.

Google-goggles does not require a cropped icon as an input in order to perform the recognition task. Nevertheless, for fairness we have tried both cropped and un-cropped images of icons. Goggles shows a low success-rate and unstable results, as shown in Figure 13.



(a) (b) (c) (d)

Figure 13. **Google-Goggles.** (a) A successful recognition of twitter. (b) Facebook icon mistakenly recognized as twitter. (c) Cut-the-rope was successfully recognized. (d) A recognition failure of a blurred Cut-the-rope icon.

5. Conclusion

In this paper we have described a practical novel solution for automatic icon scanning. Our system first detects the icon with a high success rate with no false alarms – doing all that in real-time. Next, the system recognizes the detected icon within the entire App-Store database in a few seconds. The system has been installed on several devices and tested by a number of independent users that reported high satisfaction from its performance. An embodiment is released as in app in App-Store, currently named "eyeconit".

While the current system was designed and tested for scanning App-icons, we consider it as the first step towards a complete icon-recognition system. Our future goal is to extend this and propose a more visually appealing alternative to QR Codes.

Acknowledgements: This research is supported in part by the Ollendorf foundation, the Israel Ministry of Science, and by the Israel Science Foundation under Grant 1179/11.

References

[1] QR Code is registered trademark of DENSO WAVE INCORPORATED. http://www.qrcode.com/. 1

[2] Appsnap ltd. www.getappsnap.com. 1

[3] Google ltd. - google mobile. google search app. http://www.google.com/mobile/goggles/. 1, 2, 5

[4] Mysql - database software. download.oracle.com/docs/cd/E17952_01/refman-5.1-en/mysql-indexes.html. 8

[5] Snaptell inc. acquired by a9.com. www.snaptell.com. 1

[6] Tineye mobile, ide inc - the visual search company. http://ideeinc.com/products/tineyemobile/. 2, 5

[7] A. E. Abdel-Hakim and A. A. Farag. Csift: A sift descriptor with color invariant characteristics. *CVPR (2)*, pages 1978–1983, 2006. 2

[8] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *PAMI*, 33:898–916, 2011. 2

[9] J. Canny. A Computational Approach to Edge Detection. *PAMI*, 8(6):679–698, Nov. 1986. 2

[10] O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: min-hash and tf-idf weighting. *Proceedings of the British Machine Vision Conference*, 2008. 4

[11] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973. 3

[12] J. M. Geusebroek, R. van den Boomgaard, A. W. M. Smeulders, and H. Geerts. Color invariance. *IEEE Transactions on PAMI*, 23(12):1338–1350, 2001. 4, 5

[13] H. Jegou, M. Douze, and C. Schmid. Packing bag-of-features. *The 12th ICCV*, pages 2357 –2364, 2009. 4

[14] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. *CVPR*, pages 3304–3311, 2010. 4

[15] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. *The 12th ICCV*, pages 2130 –2137, 2009. 4

[16] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004. 2, 4

[17] B. C. Russell, W. T. Freeman, A. A. Efros, J. Sivic, and A. Zisserman. Using Multiple Segmentations to Discover Objects and their Extent in Image Collections. *CVPR*, 2:1605–1614, Oct. 2006. 2

[18] J. Sklansky. Finding the convex hull of a simple polygon. *Pattern Recognition Lett*, 1:79–83, 1982. 3

[19] S. Suzuki and K. Abe. Topological structural analysis of digitized binary images by border following. *CVGIP*, 30(1):32–46, 1985. 2

[20] willowgarage. http://opencv.willowgarage.com/. 7

[21] D. Xu, C. Doutre, and P. Nasiopoulos. Correction of clipped pixels in color images. *IEEE Transactions on Visualization and Computer Graphics*, 17(3):333 –344, march 2011. 5

[22] M. Zhao, W. Zhang, Z. Wang, and F. Wang. Spatially adaptive image deblurring based on nonlocal means. *The 3rd CISP*, pages 853 –858, 2010. 5