ObjectClassifierViews: Support for Visual Programming in Papier-Mâché

Edward S. De Guzman, Ana Ramírez
Department of Electrical Engineering and Computer Science
360 Soda Hall #1776
University of California, Berkeley, CA 94720, USA
Tel: 1-510-643-3125
Email: {edwardd,anar}@eecs.berkeley.edu

ABSTRACT
A major component of camera-based user interfaces is the image classifier which takes an image as input and looks for groups of pixels having a previously specified feature such as color or location. Despite their utility in systems for image retrieval, image processing, and prototyping camera-based applications, image classifiers in these existing types of systems provide little, if any, support for viewing and modifying the parameters for calibrating the classifier. In this paper, we present ObjectClassifierViews, a family of graphical user interfaces (GUIs) for exposing to the developer the internal state of the image classifier. ObjectClassifierViews are part of the Papier-Mâché toolkit and allow the user to calibrate a classifier at runtime by directly manipulating the parameters or by providing examples of positive classifications. We present an evaluation of ObjectClassifierViews with an in-lab user study, which reveals that the added visibility of a classifier's internal state provided by the GUI allows the user to more quickly prototype image classifiers.

INTRODUCTION
Papier-Mâché [5] is a toolkit that lowers the floor for developing tangible user interfaces, especially those employing computer vision. It supports several types of input: computer vision, electronic tags, and barcodes.

When building an application with Papier-Mâché the developer must specify the input device(s) to be used and make associations between the presence or absence of physical objects (phobs) and an action. The resulting application will generate phobs by analyzing input from the capturing device and for each phob check for a matching association. If the phob has features (e.g. square, orange, RFID tag #5) that match an association, the application will perform the corresponding action as defined by the developer.

Example: In/Out Board-Controlled Media Player
Suppose you are building an In/Out Board out of a wall, construction paper and a web cam that controls the music played in an office. Based on the employees' music preferences and who is in the office, the application would select the most appropriate music (e.g., Jazz, Latin, or Pop) to be played. To build this application the developer would need to:

- specify the input device is a camera (the only input in this case)
- associate the "in" markers of the group of people who like Jazz with the Jazz radio station
- associate the "in" markers of the group of people who like Latin music with the Latin radio station
- associate the "in" markers of the group of people who like Pop music with the Pop radio station.

This application would monitor the tallies of the number of employees in the office per music genre and change styles once the majority's preferred music genre has changed.

Classifiers monitor the list of phobs seen by the system and can be customized to perform an action only when phobs having a certain feature have been recognized. For example, suppose a preference for Jazz music was represented by blue phobs. In order to find all the "In" markers for the group of people who like Jazz, a region of interest (ROI) classifier and a mean color classifier can be used. The developer customizes the ROI classifier to perform actions only on phobs in the "In" column and a mean color classifier to watch for blue phobs. When an object is added to the "In" column, it will be recognized by the ROI classifier. If it also recognized by the "blue" mean color classifier, the jazz tally is increased.

Calibrating Image Classifiers
A key part of applications that use camera input is the set of image classifiers. Making the process of authoring and calibrating classifiers easier and more robust is important

Figure 1: Screenshots of ObjectClassifierViews. From left to right: ROI Classifier, Size Classifier, Mean Color Classifier

for these types of applications. Since these types of applications are also often sensitive to lighting changes and setup changes, it is important to be able to quickly and easily re-calibrate these classifiers.

A controlled, in-lab evaluation of Papier-Mâché [5] revealed it is often difficult to specify the target feature of a classifier through text. While it may be easy to build a classifier for RFID input by specifying one or more RFID tags to look for, building image classifiers is a more difficult task to perform through text. For example, to build an ROI classifier, the developer must provide the coordinates of the top-left point of the rectangular region of interest as well as the width and height of the rectangle. Being able to specify the region by clicking and dragging with a mouse is likely to be more familiar to the developer.

In this paper we address this issue and further increase the visibility of application behavior with the introduction of ObjectClassifierViews, a family of graphical user interfaces for calibrating image classifiers. These graphical user interfaces increase the visibility of application behavior by enabling visual, real-time calibration of image classifiers.

The remainder of the paper is structured as follows: first, we will discuss the key features and implementation details of ObjectClassifierViews. Next, we discuss the results of an in-lab user study of ObjectClassifierViews. Finally, we discuss opportunities for future work in this area.

IMPLEMENTATION

To support the user's ability to visually author and calibrate image classifiers, we have designed and implemented a family of ObjectClassifierViews (see Figure 1). ObjectClassifierViews are graphical user interfaces (GUIs) which expose the internal state of a classifier to the user and allow him/her to change the state by modifying its parameters. We modified the `ObjectClassifier` class so that every instance of this object has an associated ObjectClassifierView. All ObjectClassifierViews provide the user with the same set of functionality for viewing and modifying classifiers:

- *Direct handles to the parameters* (Figure 2, area 1). The left hand side of the GUI lists the defining parameters of the classifier and shows their current values. If appropriate, the user may modify these values by changing the number in the text field.
- *Visualization of classifier parameters* (Figure 2, area 2). The in-lab evaluation of Papier-Mâché revealed that users in general did not have a very clear sense of how changes in the classifier parameters would affect the classifier's target feature (e.g., how a color with intensity 0.5 differs from a color with intensity 0.6). To address this issue, the GUI provides a visualization of the parameters below where they are displayed. For example, below the parameters specifying a color, a swatch of the color would be displayed.
- *Modification of parameters at runtime*. At the bottom left of the GUI is a button panel allowing the user to preview, undo, or accept any changes made to the classifier. This feature is useful in that changes can be seen immediately in Papier-Mâché's debugging window, resulting in a quicker feedback loop and avoiding the need to recompile the source code after each modification.
- *Two modes of calibration* (Figure 2, areas 1,4). After viewing the parameters and their visualization, the GUI offers the user two ways to modify the parameters if desired. The left half of the interface provides a direct manipulation approach, where the user can modify the parameters given that he/she has a sense of the optimal values or visualization of the values in mind. In the right half of the interface, the user can calibrate the classifier by example. This is done by first creating a list of phobs that should belong in the classifier from the list generated by Papier-Mâché. Upon clicking the "Calibrate Classifier" button, the parameters of the classifier are automatically changed to values such that the phobs listed will be classified by the classifier.

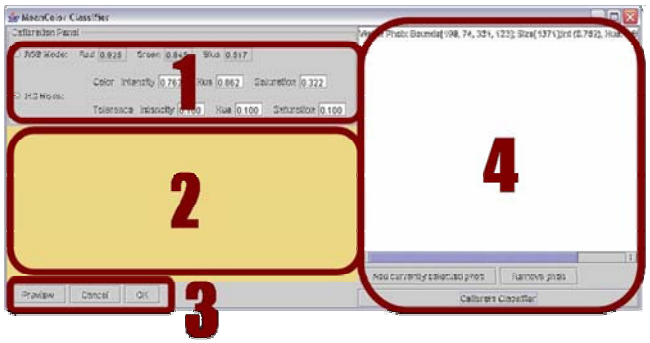

Figure 2: Screen shot of Mean Color ClassifierView with key features labeled

Currently, ObjectClassifierViews are available for building three types of classifiers: SizeClassifiers, MeanColorClassifiers, and ROIClassifiers. These classifiers and their views are discussed in further detail below (see Figure 1 for screenshots).

SizeClassifier. A SizeClassifier will classify phobs that have a size within a specified range, where "size" is defined as the sum of edges (in pixels) around the phob. The user can directly manipulate these parameters by either specifying the phob's ideal size and a variance or by specifying a range of size values. The visualization of the parameters consists of a drawing of two squares having the same sizes as the low and high points of the specified range. This gives the user an approximate estimate of what size of phobs the classifier is currently configured to classify.

Mean Color Classifier. A Mean Color Classifier will classify phobs based on the mean color of all the pixels in the phob. The left side of the MeanColorClassifierView provides handles to modify the intensity, hue, saturation, as well as the tolerance of these parameters of the mean color classifier. Red, green and blue handles are also provided in case a user prefers to work in this color space. A color swatch displaying the color represented in the parameters is also displayed on the left side of the GUI. The right side provides an interface to calibrate the classifier based on example phobs.

ROI Classifer. A Region of Interest (ROI) Classifier will classify phobs whose centers lie within one of a set of rectangular areas defined by the user. In the left side of the GUI, an area similar in size to the captured image is shown with the centers of the phobs indicated by small red squares. The user can define rectangular regions within this area using direct manipulation by clicking and dragging with the mouse. A yellow rectangle will appear around the region defined by user, and three options will then be available. The user may add this region to the current set of rectangular areas, remove this area from the set, or denote this rectangular region as the sole region of interest. Alternatively, the user can select a list of phobs on the right side of the GUI and the system will automatically define a set of regions such that the specified phobs will be classified by the classifier. The visualization of the resulting set will also appear on the left side of the GUI.

EVALUATION

We conducted an informal, within-subject, controlled evaluation of our three ObjectClassifierViews. Four students in our university's computer science department participated in the study: one undergraduate, one HCI graduate student, and two programming languages graduate students.

Each participant was asked to complete six calibration tasks: two with each classifier, one using the classifier view (GUI) and one just modifying the parameters to the classifier in the code directly (text). All the tasks were motivated by the in/out board in Figure 3. The in/out board has two columns, the "In" and "Out" column, and one row for each person. The markers in the "In" column denote the person in the corresponding row is currently in the office and the markers in the "Out" column denote the person is out. The size of the markers in the "Out" column denote how long that person will be out, while the color denotes whether the person is out on business or pleasure. The order of the tasks was varied to avoid learning effects.

The tasks to be completed using the region of interest classifier included: "Find all phobs in Aaron's row" and "Find all phobs in the "in" column". The tasks intended to be completed using the mean color classifier included: "Find all orange phobs" and "Find all green phobs." Finally, the tasks intended to be completed using the size classifier included: "Find all 15 min break segments" and "Find all 1 hour break segments."

Figure 3: Example tasks from lab study. The areas or phobs outlined represent the targets for the particular task. From left to right: "Find all phobs in Aaron's row", "Find all 15-minute breaks", "Find all green phobs"

Results

As shown in Table 1, all users were able to perform almost all the given tasks in the same time or less with the GUI when compared to modifying the parameters by text. The one exception is an instance where a user was slightly faster with text than the GUI when calibrating the SizeClassifier. The average decrease in time among users (Figure 4,5) of the MeanColorClassifierView and the ROIClassifierView is approximately 50% (55% and 48%, respectively). This data suggests that visually authoring classifiers through a

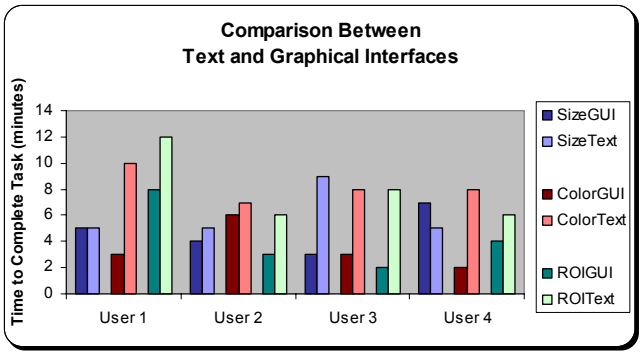

Figure 4: Time spent completing evaluation tasks

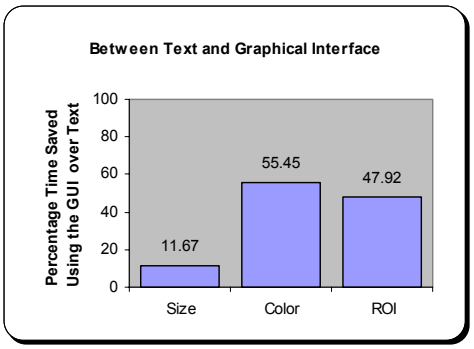

Figure 5: Percent time saved by GUI over all users

GUI is likely to require less time than constructing a classifier through text.

During our exit interviews, we learned that users in general preferred to build classifiers through means other than specifying numerical values for the parameters. When using the GUI, all users preferred to calibrate the classifier either by example (Size and MeanColor) or with the mouse (to specify an ROI) as opposed to reading the numerical data produced by Papier-Mâché and inputting them to the provided text fields. However, the users felt that the numerical values should still be visible and editable so that the user can decide which approach would work best for him/her. When asked to describe the process of accomplishing the task when the GUI was available, one user would calibrate by example as a first pass at completing the task. He would then modify the text fields for fine tuning the parameters if it was necessary.

Task	User1	User2	User3	User4	Avg. Diff
Size GUI	5	4	3	7	
Size Text	5	5	9	5	
Diff	0	20%	66.7%	-40%	11.7%
Color GUI	3	6	3	2	
Color Text	10	7	8	8	
Diff	70%	14.3%	62.5%	75%	55.4%
ROI GUI	8	3	2	4	
ROI Text	12	6	8	6	
Diff	33.3%	50%	75%	33.3%	47.9%

Table 1: Time (in minutes) spent by users on the evaluation tasks and the percent time saved by the GUI

RELATED WORK

Several researchers have applied image classifiers to the task of querying images by their content [2,3,4,6]. In these systems, the entire image is queried by searching for a group of pixels having a pattern similar to the search item. BlobWorld [1] takes a step towards a more efficient search by first segmenting the image to be searched into smaller regions known as "Blobs" and then attempting to match the search terms in the generated list of blobs for each image. This approach is similar to Papier-Mache's method of searching for features in segmented phobs as opposed to the entire captured image.

Most similar to this work is the Crayons system. Crayons is a tool for creating image classifiers that can be integrated into Java programs. The image classifiers use pixel-color features to classify images. Users can build classifiers without possessing detailed knowledge of image processing by drawing on training images to indicate what regions of the image are to be recognized. This is similar to our "calibration by example" approach to building classifiers, where the designer provides examples of correct classifications. However, unlike Crayons, our augmentation to the Papier-Mâché toolkit with ObjectClassifierViews does not employ machine learning techniques. The SizeClassifier and ROIClassifier do not limit the user to building classifiers which only recognize pixel-based features. In addition, ObjectClassifierViews expose the parameters to their associated classifier. By allowing the user to choose between direct manipulation of parameters and calibration by example, we support designers of varying levels of experience with image processing. Experts with a better sense for the correct parameter values for a classifier may choose to directly modify them while novices may wish to specify them with examples. While Crayons is a stand-alone tool for creating classifiers, ObjectClassifierViews are an end-user programming module within a larger toolkit. As a result, designers can not only build classifiers but also specify how an application should behave when a phob is recognized by a particular classifier.

CONCLUSION AND FUTURE WORK

We have presented ObjectClassifierViews, a family of GUIs allowing designers to visually author image

classifiers for the Papier-Mâché toolkit. These GUIs give the user a direct handle to view and manipulate an image classifier's parameters at runtime. In addition, they give the user the option of specifying parameters by directly manipulating numerical values or by providing examples, thus supporting users of various backgrounds in computer vision and image processing. A user study showed that image classifiers can be authored more quickly with a GUI than by modifying source code. Exit interviews with user study participants revealed that authoring classifiers by example is preferred over specifying numerical values.

The future work on this project includes refinements on the GUIs for calibrating the classifiers based on user feedback. The classifiers in Papier-Mâché can now be calibrated using GUIs, but the new values cannot currently be saved. We leave the serialization of the new classifier values to future work. Papier-Mâché's monitor window displays which objects fall within a classifier, but it does not give any information about how close to the "edge" of the classifier they are. We think it might also be useful to see which objects fall outside a classifier and by how much.

In addition to iterating over our current ObjectClassifierViews, we are also exploring other instances within Papier-Mâché where tasks which currently can only be accomplished textually can be performed more efficiently using a graphical user interface.

ACKNOWLEDGEMENTS

We would like to thank Scott Klemmer and Jack Li for their enthusiasm and feedback on the project and for the use of Scott Klemmer's toolkit, Papier-Mâché. We would also like to thank the CS294-2 (Ubiquitous Computing Seminar) class for their feedback during the in-class project critique.

REFERENCES

1. Carson, C., Thomas, M., Belongie, S., Hellerstein, J.M. and Malik, J. "Blobworld: A system for region-based image indexing and retrieval." In Proc. Int. Conf. Visual Inf. Sys., 1999.
2. Christos Faloutsos, C., Barber, R., Flickner, M., Niblack, W., Petkovic, D. and Equitz, W. "Efficient and effective querying by image content." Journal of Intelligent Information Systems, 3(3/4): 231-262, July 1994
3. Hirata, K. and Kato, T. "Query by Visual Example— Content-Based Image Retrieval." Advances in Database Technology EDBT '92, 3rd International Conference on Extending Database Technology, Vienna, Austria, A. Pirotte, C. Delobel, and G. Gottlob, eds., Lecture Notes in Computer Science, vol. 580, Springer-Verlag, Berlin, 1992, pp. 56-71.
4. Kelly, P. M. and Cannon, M. "Query by Image Example: the CANDID Approach." Los Alamos National Laboratory White Paper, (1995).
5. Klemmer, S., Li, J., Lin, J., and Landay, J., "Papier-Mâché: Toolkit Support for Tangible Input." CHI Letters, Human Factors in Computing Systems: CHI2004. 6(1).
6. Vailaya, A., Zhong, Y., and Jain, A. K. "A hierarchical system for efficient image retrieval." In Proc. Int. Conf. on Patt. Recog. (August 1996).