A Design Tool for Camera-based Interaction

Jerry Alan Fails, Dan R. Olsen
Computer Science Department
Brigham Young University
Provo, Utah 84602
{failsj, olsen}@cs.byu.edu

ABSTRACT
Cameras provide an appealing new input medium for interaction. The creation of camera-based interfaces is outside the skill-set of most programmers and completely beyond the skills of most interface designers. Image Processing with Crayons is a tool for creating new camera-based interfaces using a simple painting metaphor. A transparent layers model is used to present the designer with all of the necessary information. Traditional machine learning algorithms have been modified to accommodate the rapid response time required of an interactive design tool.

Categories & Subject Descriptors: H.1.2 [Information Systems]: User/Machine Systems — Human factors; H.5.2 [Information Interfaces and Presentation]: User Interfaces — interaction styles, prototyping, theory and methods; I.4.8 [Image Processing and Computer Vision]: Scene Analysis — color, object recognition, tracking; I.4.9 [Image Processing and Computer Vision]: Applications; **General Terms**: Algorithms, Human Factors; **Keywords**: Image processing, classification, interaction, machine learning, perceptive user interfaces.

INTRODUCTION
Myron Krueger's VideoPlace [20] established cameras as an interactive input medium and sparked an interest in user interfaces that function by watching what people do rather than requiring overt inputs by a user. Mark Weiser popularized the notion of ubiquitous computing [32] where computing moved into the environment as a whole rather than residing on a desk. Hiroshi Ishii and his Tangible Media Group [14,28,29] have pioneered demonstrations of how computing and interaction can be embedded in everyday things. Achieving these visions requires a much larger range of interactive sensors than can be found on common desktop systems.

In this work we are interested particularly in camera-based interfaces. We chose the camera because of its ambient nature in that it passively watches without requiring the user to wear or carry anything special. Cameras are currently quite cheap with their costs rapidly approaching the cost of a keyboard and mouse. Projects such as the Gesture Pendant [26], Light Widgets [9], finger tracking [6], hand waving [11] and Laser Pointer Interaction [22] have demonstrated how cameras can form the primary sensor for an interactive behavior.

The challenge, however, is that designing camera-based interfaces is quite difficult. Not only is sophisticated programming required, but also the mathematics of image processing and machine learning. These are far beyond the skill set required for Visual Basic. In our Image Processing with Crayons (Crayons) project we are focused on creating interface development tools that can be placed in the hands of ordinary designers. Since there is no extant design community for camera-based interfaces we have made several assumptions about the skill set such a community might posses.

- Familiarity with visual design tools like Visual Basic, and how they integrate with other software,
- No familiarity with image processing or machine learning,
- An understanding that image regions must be classified to identify the items of interest to the interaction, but little understanding of how that might be done.

At the heart of any camera-based interaction is a classifier that takes an image and identifies those pixels or groups of pixels that are of interest to the interaction. This is so because once a programmer knows where items of interest are found in an image, the remaining programming is very similar to mouse-based interaction. The classifier is the particular part of camera-based interaction that most programmers shy away from.

Because UI designers rarely have detailed image processing or machine learning knowledge, they must learn these skills before even beginning to build such a classifier. Once the knowledge of how to build the classifier has been attained the implementation is not usually difficult, although there are still many tricks and nuances that require repetitive testing and alterations. Kernels, filters and machine learning algorithm parameters can be tricky and temperamental and generally require many modifications. In our experience, adjustments to a manually created

CHI 2003, April 5-10, 2003, Ft. Lauderdale, Florida, USA.

classifier may take weeks, if not months to finally achieve satisfactory results.

It is important to note that image classifiers can be difficult for even knowledgeable programmers to get right. In the traditional approach, a programmer must identify a set of features that are sufficient for the task. Most machine learning algorithms assume that substantial time and thought will go into feature selection before the classifier algorithm is even invoked. Such features may be as simple as the hue and saturation values for a particular pixel or more complicated convolution kernels for separating desired pixels from similar ones. A programmer can spend months researching, programming and trying different combinations of features. Such activities are quite beyond the skills of the average interface designer. However, understanding the interaction problem is well within the capacities of an interface designer.

Consider the example of laser-pointer interaction. Once the laser spot is accurately located the remaining interactions are very similar to mouse interactions. Anyone looking at the camera images can readily identify the key problem as "find that red spot." Finding the red spot, however, is not trivial due to the laser overdriving the camera, other red objects in the field of view, and noise from cheap cameras.

Similar problems arise when using hands as the input. Techniques such as Light Widgets [9] are founded on skin tracking to locate user's hands relative to various input areas. Any person can look at the image and locate the hands, however, the algorithms that create classifiers and take care of noise and other artifacts take substantial time to design.

Crayons is a tool that focuses on creating such classifiers and exporting them in a form that is easily incorporated into Java programs. Crayons is intended to aid UI designers who do not have detailed knowledge of image processing and machine learning. It is also intended to greatly accelerate the efforts of knowledgeable programmers.

Crayons currently uses pixel-color features to classify. It uses pixel value sums for neighboring and concentric rectangular regions. Many other feature types have and are used in other systems: pixels, texture, shape, region, motion, history, whole images, etc. Although we hope Crayons eventually incorporates more feature types, the current implementation only uses pixel features. These features are unable to distinguish shapes and object orientations like hand gestures, but do well with pixel-based object detection like laser, hand and object tracking.

Figure 1 – Classifier Design Process

The Crayons interface design process is shown in Figure 1. Designing an interface with Crayons begins with gathering a collection of representative images of situations where the interaction will occur. Because Crayons classifiers are trained, the selection of the images on which the training is to be done is important. For example if the technique is to work in a variety of lighting conditions, images with such variety must be chosen. If one were creating a skin tracker, images with a variety of skin colors are needed. The sample image selection step requires some consideration since the training images affect the final classifier. Although initial thought is beneficial, it is not required because new images can always be inserted into the Crayons to aid classifier creation.

After collecting sample images, the designer uses the Crayons tool to train a classifier. The training metaphor is to simply paint the classification over the sample images. If while using a classifier a situation is discovered where it does not work correctly, new training images can be added and the designer can paint corrections on them. The total number of images needed depends on its targeted application. When the designer has a classifier that is satisfactory, the classifier is exported as a serialized Java object that can be easily loaded into any Java program.

There are two primary goals for the Crayons tool: 1) to allow the user to create a classifier quickly, and 2) to allow the user to focus on the classification problem rather than image processing or algorithms. Crayons is successful if it takes minutes rather than weeks or months to create an effective classifier. For simplicity sake, we will refer to this as the UI principle of *fast and focused* — the fundamental UI principle of enabling the designer to quickly accomplish his/her task while remaining focused solely on that task.

Traditional machine learning does not directly solve the Crayons problem. We found in the course of this work that traditional machine learning makes assumptions about the nature of the learning problem that are not valid in the Crayons approach. These assumptions and their resolution will be discussed later in the paper.

Crayons Overview

The overall purpose of Crayons is to create classifiers easily and rapidly. Crayons receives images upon which the user does some manual classification, a classifier is created, then feedback is displayed. The user can then refine the classifier by adding more manual classification, or, if the classifier is satisfactory, the user can export the classifier.

Although the focus of this paper is on the internal processes of Crayons, its input and output deserve some attention. For the images (the input arrow) the system assumes the user has already taken digital pictures and saved them as files to import into the system, or that a camera is set up on the machine running Crayons, so it can capture images from it. Additional images can always be

inserted to enhance training and testing. Exporting the classifier (the output arrow) is trivial. Since our implementation is written in Java, our classifier is simply serialized.

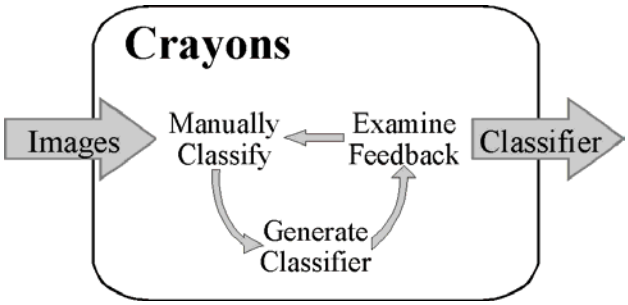

Figure 2 – The classification design loop

The most interesting issues with the development of Crayons lie in the interactive loop displayed in Figure 2. To accomplish the *fast and focused* UI principle, this loop must be easy and quick to cycle through. The cycle can be broken down into two components: the UI and the Classifier. The UI component needs to be simple so the user can remain focused on the classification problem at hand. The classifier creation needs to be efficient so the user gets feedback as quickly as possible, so they are not distracted from the classification problem.

USER INTERFACE

In designing a classifier interface there are four pieces of information that the designer must be able to manipulate.

1. The set of classes or "crayons" to be recognized,
2. The set of training images to be used and in particular, the current training image that the designer is trying to classify,
3. The classification of pixels as defined by the designer (the manual classification),
4. The classifier's current classification of the pixels (the feedback).

The primary metaphor for Crayons is painting with user-defined classes as "crayons". [To emphasize the paint metaphor, we use the verb "paint" rather than the traditional "color" with crayons.] The designer can create as many crayons as desired, associate a color to that crayon, and then paint with them over training images in the same fashion as a typical paint program.

Suppose we are trying to create a hand tracker [21, 33]. Figure 3 shows the control interface that allows the designer to create classes. For the hand tracker we have two output classes or crayons: Skin and Background. The Crayons tool always provides a "Nothing" crayon, which is completely transparent. The paint layer is initialized to "Nothing" for every pixel. The "Nothing" crayon can also be used to undo or erase errant manual classifications. As previously stated, the designer can create as many classes or crayons as desired by giving each a name and an associated color.

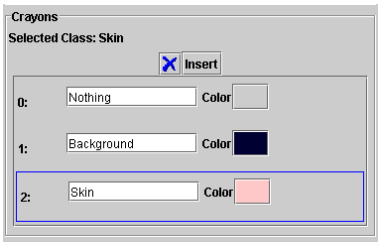

Figure 3 – Creating Class Crayons

In order to understand the classification process, the designer needs to see the underlying image, the pixels classified by the designer, and the pixel classification of the currently trained classifier. This information is provided in three layers where the training class layer and classifier feedback layer are semi-transparent. All three of these are layered on top of each other, so the user can easily see how they coincide. This helps the user remain focused on the task, and constantly reminds the user of the goal to have a correctly classifying classifier.

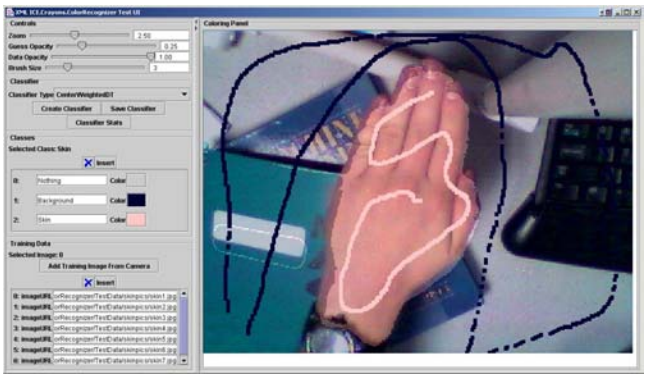

Figure 4 – The Crayons User Interface

Figure 4 shows the designer having painted some Background in a dark blue and some Skin in a light pink. Note that the user does not need to color the entire image. By coloring only a few pixels as in Figure 4, the classifier then generates its best classification for all of the pixels in the image. In Figure 4, the classifier feedback is quite transparent and therefore hard to see unless you know what you are looking for. We resolve this by allowing the user to control the opacity of the paint layer and the classifier feedback layer. The user may want to see more of the underlying image, to paint with the crayons, or perhaps, they would just like to get an overall idea of what the classifier looks like.

Figure 5 – Varying layer opacity

In the left-most image of Figure 5 it is easy to see the training paint but the classification is obscure. In the center image, paint and classification are all visible and one can still see that the edges of the hand are not correctly

classified. On the right where the classifier layer is opaque we lose information about the underlying hand image, but it is apparent that to the left of the hand, the classifier is making many small mistakes. The designer can correct those mistakes by painting the noise to the left of the hand with the Background crayon. By zooming in, the designer can correct the edges of the hand by more carefully painting with the Skin crayon.

The classifier design process then is to define the desired crayons (output classes), quickly paint samples of those classes onto one or more training images and then request a generated classifier. The designer then views the classification and paints corrections as necessary where the classifier was incorrect. Once the first image is classifying acceptably, the designer flips to the next image and paints any necessary corrections. As the classifier improves and is confronted with new training data it will change its behavior. By reviewing all of the training images and painting corrections, the trained classifier will quickly converge, provided there are sufficient features to perform the classification. The process can be summarized as a paint, view, correct loop.

The following sequence of images shows the process of paint, view, and correct.

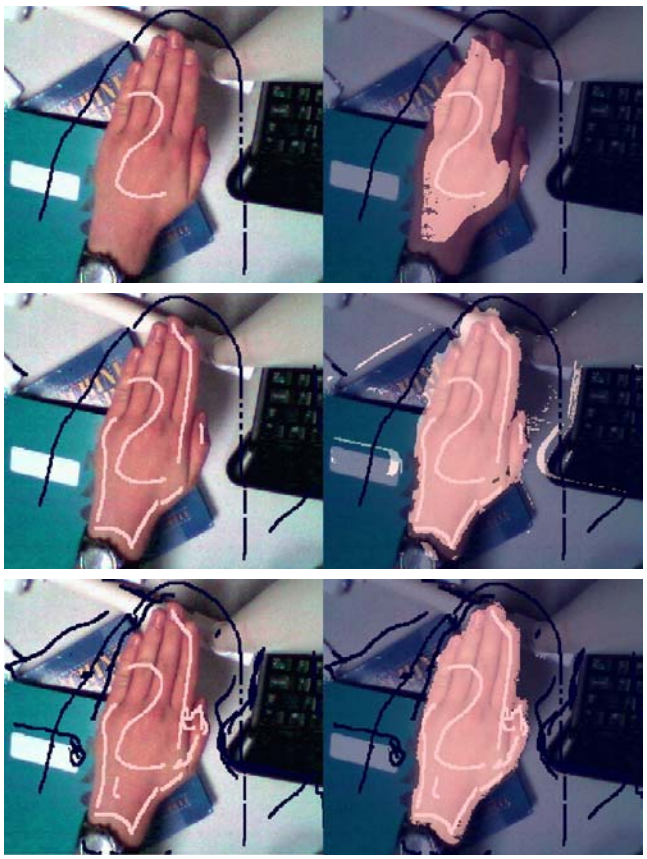

Figure 6 – Crayons interaction process: paint, view, correct

Figure 6 illustrates how the user initially paints very little data, views the feedback provided by the resulting classifier, corrects by painting additional pixels and then iterates through the cycle. As seen in the topmost image pair in Figure 6, only a little data can generate a classifier that roughly learns skin and background. The classifier, however, first over-generalizes in favor of background; so, in the middle image pair you can see skin has been painted where the classifier did poorly at classifying skin. The resulting classifier shown on the right of the middle image pair shows how the new classifier classifies the most of the skin on the hand, but is now classifying some of the background as skin. The classifier is corrected again, and the resulting classifier is shown as the final image pair in the sequence. Thus, in only a few iterations, a skin classifier is created.

CRAYONS CLASSIFIER

The automated classifier creation is the brain of the Crayons tool. It needs to be able to extract features and generate a classifier quickly to accomplish the "fast" in the *fast and focused* principle. Classical machine learning generally has the following assumptions:

- There is a relatively small, fixed set of carefully chosen features,
- There is limited training data,
- The classifier must amplify that limited training data into excellent performance on new training data,
- Time to train the classifier is relatively unimportant as long as it does not take too many days.

None of the above assumptions hold in our interactive situation. UI designers do not need to know which features are appropriate. Crayons insulates them from having to know such things. In our current Crayons prototype there are about 175 features per pixel [Features will further be explained in their own subsection.] To reach the breadth of application that we desire for this tool we project over 1,000 features will be necessary. For any given problem somewhere between three and fifteen of those features will actually be used, but the classifier algorithm must automatically make this selection. The classifier we choose must therefore be able to accommodate such a large number of features, and/or select only the best features.

When a designer begins to paint with Crayons on an image a very large number of training examples can quickly be generated. With 77K pixels per image and 20 images, one can rapidly generate over a million training examples. In practice, the number stays in the 100K example range because designers only paint the pixels that they need to correct rather than all pixels in the image. What this means, however, is that designers can generate a huge amount of training data very quickly. Traditional machine learning generally focuses on the ability of a classifier to predict correct behavior on new data. Our experience is that interactive classifier training is distinctly different from batch-oriented machine learning test beds. If the classifier's predictions for new data are wrong, the designer can

rapidly make appropriate corrections. By rapid feedback and correction the classifier is quickly (in a matter of minutes) focused onto the desired behavior. The goal of the classifier is not to *predict* the designer's intent into new situations but rapidly *reflect* that intent as expressed in concrete examples.

The interactive classification loop requires that the classifier training be fast. To be effective, the classifier must be generated from the training examples in 1-3 seconds. If the classifier takes minutes or hours, the process of 'paint-view-correct' is no longer interactive, and is much less effective as a design tool. Training on 200,000 examples with 150 features each in less than 3 seconds is a serious challenge for most machine learning algorithms.

Lastly, for this tool to be viable the final classifier will need to be able to classify 320 x 240 images in less than a fourth of a second. If the resulting classifier is much slower than this it becomes impossible to use it to track interactive behavior in a meaningful way.

With these assumptions in mind we must resolve which machine-learning algorithm to use to create our classifier and which features to extract from the training data.

Machine Learning

We require a machine-learning algorithm that can handle a large number of examples with a large number of features. The algorithm must rapidly discard irrelevant features and produce an efficient resulting classifier. We discuss several options and the reason why they are not viable before settling on our algorithm of choice: decision trees.

Neural Networks [24] are a powerful and often used machine-learning algorithm. The strength of neural networks lies in their ability to approximate higher order decision boundaries, their compactness, and their speed in classification. Despite these advantages, they also have serious drawbacks. The intrinsic fault of this algorithm is that it will take far too long for its training to converge — much too long to be interactive. Another drawback results from the exponential number of feature combinations that must be learned. The exponential number depends on the amount of features and hidden nodes. This curse of dimensionality problem is insurmountable because of the number of features that IML and Crayons require.

The nearest-neighbor algorithm [5] is easy to train but would not be effective in Crayons for a few key reasons. First, nearest-neighbor does not discriminate among features. Second it has serious problems in high dimensional feature spaces of the kind needed in Crayons. And lastly, nearest-neighbor generally has a classification time that is linear in the number of training examples which makes it completely unacceptable.

There are yet other ensemble techniques such as boosting and classifier cascading that do well with feature selection, which is a desirable characteristic. However, while boosting has shown itself to be very effective on tasks such as face tracing [31], its lengthy training time is prohibitive for interactive use in Crayons. These ensemble techniques may be possible in an offline optimization of the classifier, but are infeasible in an interactive session.

There are many more machine-learning algorithms, however, this discussion is sufficient to preface to our use of decision trees. All the algorithms discussed above suffer from the curse of dimensionality. When many features are used (100s to 1000s), their creation and execution times dramatically increase. In addition, the number of training examples required to adequately cover such high dimensional feature spaces would far exceed what designers can produce. With just one decision per feature the size of the example set must approach 2^{100}, which is completely unacceptable. We need a classifier that rapidly discards features and focuses on the 1-10 features that characterize a particular problem.

Decision trees [23] have many appealing properties that coincide with the needs of Crayons. First and foremost is that the decision tree algorithm is fundamentally a process of feature selection. The algorithm operates by examining each feature and selecting a decision point for dividing the range of that feature. It then computes the "impurity" of the result of dividing the training examples at that decision point. One can think of impurity as measuring the amount of confusion in a given set. A set of examples that all belong to one class would be pure (zero impurity). There are a variety of possible impurity measures [8]. The feature whose partition yields the least impurity is the one chosen, the set is divided and the algorithm applied recursively to the divided subsets. Features that do not provide discrimination between classes are quickly discarded. The simplicity of decision trees also provides many implementation advantages in terms of speed and space of the resulting classifier.

Quinlan's original decision tree algorithm [23] worked only on features that were discrete (a small number of choices). Our image features do not have that property. Most of our features are continuous real values. Fayyad and Irani [10] have shown how decision trees can be built using threshold values. By selecting a threshold T for a given feature F we can divide the training examples into two sets where $F<T$ and $F>=T$. The trick is for each feature to select a value T that gives the lowest impurity (best classification improvement). The selection of T from a large number of features and a large number of training examples is very slow to do correctly. Even an O(N log N) algorithm is prohibitively slow for our interactive use.

Our experience with Crayons has shown that selecting a correct threshold T for a given feature is not as important as selecting one that partitions the set of examples into two roughly equal sized subsets. If the subsets are relatively equal, the height of the decision tree is always close to $\log_2 N$, which is highly efficient, even for very large N. We

therefore only look at the middle twenty percent of the values for a given feature and find the *T* that has the least impurity for each feature *F* and compare the impurities of the divisions of all features. This is very efficient and produces relatively shallow decision trees. The resulting decision trees yield comparable results while being much more shallow than if all possible *T* values were tested. Designers make up for the lower quality of the decision tree with the ability to correct more rapidly. Though the learning ability of the classifier is lower, designers are able to train a good classifier much faster. The key is in optimizing designer judgment rather than classifier predictions.

If desired, after a classifier is trained, both the training data painted by the user and the resulting classifications that the user has approved can be used offline to generate more efficient and robust classifiers. Such offline approaches can take whatever time is needed and apply many traditional machine-learning approaches that are not interactively effective.

Features

As previously stated many different features have been used in the past to create visual classifiers. Crayons requires a feature set that can be combined to model a variety of classifiers and image filters. Rather than carefully select features, and various convolution filters, we let the decision tree algorithm form those filters by its own tree structure. In other words, the DT structure used in Crayons will automatically combine basic features to simulate filters and kernels.

The basic features for a given pixel are the red, green, blue, hue, saturation, and value numbers for that pixel. A number of interactive techniques can be built from these six values, but they are not very robust to noise and are not sufficient for edge detection, differentiating spots from larger areas, and a variety of other useful behaviors.

We introduced a wider range of features by computing the integral image [31] of each of these six features. The integral image is a technique for computing the sum of the values in any rectangle of an image in constant time. This technique allows us to cheaply sample multiple areas of differing sizes around each pixel. This is only possible because of the constant time summation of these regions that is afforded by integral images. These region features have been very effective in producing efficient face trackers [31]. By combining such area features the decision tree can emulate most noise filters, edge detectors, speckle removers and other image processing techniques. However, because the decision tree is doing this automatically the designer does not need to know, nor care about the existence of such filters.

The areas used as features are shown in Figure 7. For each of the 6 basic features we add the surrounding 8 pixels. We then add an area feature for that 3x3 region and the surrounding 8, 3x3 regions. We then do the same thing at a 9x9 scale and finish with the 27x27 area as a whole. Combinations of these features can produce a variety of filters at a variety of scales. At present, we are using a total of around 175 features. These features perform very well for a wide variety of interactive tracking needs.

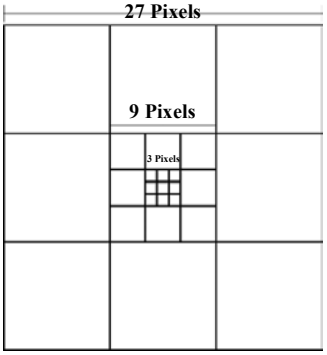

Figure 7 – Area feature pattern

There are limitations to the interactive imaging tasks that Crayons can support. The current feature set cannot take into account image change over time or motion [7]. There are well known techniques for doing this, but the generalized feature set immediately grows to thousands. This is not an algorithmic problem, but it is a performance problem on our 2.5Ghz workstations. Our future work includes features for texture recognition, blob detection and image moments of blobs as features.

EVALUATION

Our original intent was to build a tool that would allow designers to build camera-based interfaces in minutes rather than months. To evaluate this we collected several interactive problems that we have previously solved using traditional programming techniques. These include tracking a laser spot, tracking skin, tracking paper cards on a table surface and tracking a robot car driving around on the carpet.

For each of these problems we assembled a set of training images and carefully hand classified each pixel into its appropriate class for each problem. This provided us with a "gold standard" that we could use in evaluations. We then built a special version of Crayons that takes each classifier that the designer generates and compares its classifications to the manually classified gold standard and reports a percentage of pixels that are classified the same. We then gave each problem to briefly trained users and had them train a classifier until it was 98% accurate relative to the gold standard. We timed how long it took for a designer to solve each problem to get a measure of the effectiveness of the Crayons tool.

As can be seen in Table 1, which shows the results of this testing, all users were able to create a valid classifier for each of the problems in well under ten minutes. Although, we only tried this on five different users, which is not enough to show statistical significance, it is apparent that

the classifier creation time is going to be at least three orders of magnitude better than a hand-programmed approach, which could take days, weeks or months.

	Time (min:sec)		
Problem	**Min**	**Max**	**Avg**
Skin	3:28	7:00	5:20
Paper Cards	1:25	3:35	2:30
Car	0:27	1:28	1:02
Laser	0:11	1:56	0:39

Table 1 – Time spent creating classifier

While testing we observed that those who manually classified a little, created a classifier, then iterated through the interactive loop illustrated in Figure 2, were able to create a satisfactory classifier more quickly than those who laboriously added many pixels and training examples between iterations. Those who tried to out-guess the classifier generation took longer than those who let Crayons do its work. Another observation is that because of the simplicity of the interface, some people were haphazard in their manual classifications. This did not hinder them from getting a satisfactory classifier, but quite possibly could. One last observation was that most of the users (four of the five) said that their main focus while creating the classifier was to see which examples were classifying incorrectly and correctly. This was desired, as it fulfills the *fast and focused* principle discussed earlier.

RELATED WORK

Image classification is an extremely difficult problem that has been approached from many different perspectives. Much research has been done in the area of automatic image classification and indexing. Most of the research has been in the area of query by image content [4,13,18,30]. Query by image content focuses on classifying the whole image and being able to retrieve images that are similar to a query image. It is a retrieval problem, but is very similar to the classification problem that we face. Other systems like Blobworld [1,3] attempt to enhance the retrieval of images by using more localized information like local shapes and textures. These localized shape and texture features are more similar to what we do with Crayons, by using the area color features.

There are some people who have also dipped into interactive machine learning. Takagi discusses in his survey of interactive evolutionary computing (IEC) how different EC applications strive to exploit human evaluation skills [27]. Scott, et al. [25] also discuss the optimization possibilities that a human "in-the-loop" can provide.

Of all the research that has been done, the most similar to our work is the research done by Jaimes and Chang with the *Visual Apprentice*. Their research has produced a visual tool to help create hierarchical classifiers to find specific scene types in video [16,17], as well as a defining hierarchy of how to classify images in general [15]. Their work is significant, because it contains many of the ideas we are exploring, with the distinction that it does not have interactive classifier feedback. We believe that by using quickly generated classifiers, and providing rapid feedback, the user can iterate through the classifier tuning process at interactive speeds. This tightens the loop between the designer and the machine learning and enables the user to create a satisfactory classifier within minutes instead of hours, weeks, or months.

CONCLUSION

The Crayons tool is a first step in putting sensor-based interactions into the hands of user interface designers. Machine-learning for interactive use is significantly different from traditional batch-oriented training and carries its own unique demands on the algorithms. Crayons illustrates how IML can be used to free designers from the knowledge of machine learning and image/signal processing, and instead, allow them to focus on interactive tasks.

REFERENCES

1. Belongie, S., Carson, C., Greenspan, H. and Malik, J. "Color and texture-based image segmentation using EM and its application to content-based image retrieval." In Proc. Int. Conf. Comp. Vis., 1998.
2. Buck, I., Finkelstein, A., Jacobs, C., Klein, A., Salesin, D. H., Seims, J., Szeliski, R., and Toyama, K. "Performance-driven hand-drawn animation." *Proceedings of NPAR 2000* (June 2000).
3. 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.
4. 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.
5. Cover, T., and Hart, P. "Nearest Neighbor Pattern Classification." *IEEE Transactions on Information Theory*, 13, (1967) 21-27.
6. Crowley, J.L, Bérard, F., and Coutaz, J. "Finger Tracking as an Input Device for Augmented Reality." IWAGFR '95: Zurich, Germany (June 1995).
7. Davis, J. "Recognizing movement using motion histograms." MIT Media lab Technical Report #487, March 1999.
8. Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern Classification*. (2001).
9. Fails, J.A., Olsen, D.R. "LightWidgets: Interacting in Everyday Spaces." *Proceedings of IUI '02* (San Francisco CA, January 2002).
10. Fayyad, U.M. and Irani, K. B. "On the Handling of Continuous-valued Attributes in Decision Tree Generation." *Machine Learning*, 8, 87-102,(1992).

11. Freeman, W., Anderson, D., Beardsley, P., et al. "Computer vision for interactive computer graphics." IEEE Computer Graphics and Applications, Vol. 18, Num 3, pages 42-53, May-June 1998.
12. Ghidary, S.S. Nakata, Y., Takamori, T. and Hattori, M. "Head and Face Detection at Indoor Environment by Home Robot." *Proceedings of ICEE200* (Iran, May 2000).
13. 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.
14. Ishii, H., and Ullmer, B. "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms." *Human Factors in Computing Systems (CHI '97).* (March 1997).
15. Jaimes, A. and Chang, S.-F. "A Conceptual Framework for Indexing Visual Information at Multiple Levels." IS&T/SPIE Internet Imaging 2000, (San Jose CA, January 2000).
16. Jaimes, A. and Chang, S.-F. "Automatic Selection of Visual Features and Classifier." *Storage and Retrieval for Image and Video Databases VIII*, IS&T/SPIE (San Jose CA, January 2000).
17. Jaimes, A. and Chang, S.-F. "Integrating Multiple Classifiers in Visual Object Detectors Learned from User Input." Invited paper, session on Image and Video Databases, 4th Asian Conference on Computer Vision (ACCV 2000), Taipei, Taiwan, January 8-11, 2000.
18. Kelly, P. M. and Cannon, M. "Query by Image Example: the CANDID Approach." Los Alamos National Laboratory White Paper, (1995).
19. Koller, D., Klinker, G., Rose, E., Breen, D., Whitaker, R. and Tuceryan, M. "Real-time Vision-based Camera Tracking for Augmented Reality Applications." *Proceedings of the Symposium on Virtual Reality Software and Technology* (VRST-97), Lusanne, Switzerland, Sept 1997, 87-94.
20. Krueger, M. W., Gionfriddo. T., and Hinrichsen, K., "VIDEOPLACE -- an artificial reality". *Human Factors in Computing Systems, CHI '85 Conference Proceedings*, ACM Press, 1985, 35-40.
21. Hardenberg, C. and Bérard, F. "Bare-Hand Human-Computer Interaction." *Proceedings of the ACM Workshop on Perceptive User Interfaces* (Orlando, FL, Nov. 15-16, 2001).
22. Olsen, D.R., Nielsen, T. "Laser Pointer Interaction." *Proceedings of CHI '01* (Seattle WA, March 2001).
23. Quinlan, J. R. "Induction of Decision Trees." *Machine Learning*, 1(1); 81-106, (1986).
24. Rumelhart, D., Widrow, B., and Lehr, M. "The Basic Ideas in Neural Networks." *Communications of the ACM*, 37(3), (1994), pp 87-92.
25. Scott, S., Lesh, N., Klau, G. "Investigating Human-Computer Optimization." *Proceedings of CHI '02* (Minneapolis MN, April 2002), pp. 155-162.
26. Starner, T., Auxier, J. and Ashbrook, D. "The Gesture Pendant: A Self-illuminating, Wearable, Infrared Computer Vision System for Home Automation Control and Medical Monitoring." *International Symposium on Wearable Computing* (Atlanta GA, October 2000).
27. Takagi, H. "Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation." Proceedings of the IEEE, Vol. 89, Issue 9, Sep 2001, pp. 1275-1296.
28. Underkoffler, J., Ullmer, B. and Ishii, H. "Emancipated Pixels: Real-World Graphics in the Luminous Room." Proceedings of SIGGRAPH '99 (Los Angeles CA, 1999), ACM Press, 385-392.
29. Underkoffler, J. and Ishii H. "Illuminating Light: An Optical Design Tool with a Luminous-Tangible Interface." Proceedings of CHI '98 (Los Angeles CA, April 1998).
30. Vailaya, A., Zhong, Y., and Jain, A. K. "A hierarchical system for efficient image retrieval." In Proc. Int. Conf. on Patt. Recog. (August 1996).
31. Viola, P. and Jones, M. "Robust real-time object detection." Technical Report 2001/01, Compaq CRL, February 2001.
32. Weiser, M. "The Computer for the 21st Century." *Scientific American*, 1991, 365(3), pp. 94-104.
33. Yang, M.H. and Ahuja, N. "Gaussian Mixture Model for Human Skin Color and Its Application in Image and Video Databases." Proceedings of SPIE '99 (San Jose CA, Jan 1999), 458-466.