EE368 Project: Paintings at an Exhibition

Travis Skare
travissk@stanford.edu

***Abstract*—Automatic entity detection in photographs has interesting applications for consumer technology: faces can be recognized and tagged automatically without user effort. Alternatively, items of interests can be tagged and linked to an information store or location for descriptions and further information.**

For this project, a system was designed to detect the primary painting in an image taken at an art gallery. This painting is extracted and compared against a database of training images to identify the title of the image in question.

This paper describes the approach used to extract and identify these paintings.

I. Introduction

GUESTS at a museum may take pictures of art and later forget which pieces are in a picture. We show that image processing techniques can be used to identify paintings within seconds on personal computers.

Such technology has immediate and related applications such as auto-tagging friends and family in a photo, saving minutes per imported image.

This paper describes a system designed for EE368 to automatically identify one of 33 known paintings in an image taken by a relatively advanced consumer-level cell phone camera. The various steps of the algorithm are described, and an example painting is shown at various stages in the algorithm.

The system consists of two stages: painting extraction and painting identification. The painting extraction phase accepts an RGB image of any size and identifies the primary painting in the image. It crops the picture to the painting, adjusts perspective and rotation distortions and returns an RGB image of the painting as viewed from the front.

The identification phase accepts this RGB image, converts it to a grayscale image and uses an "Eigenpainting" approach to identify the painting against a known training set. This phase borrows from Turk and Pentland's use of Eigenfaces for face recognition, which is described in [2].

This project was coded exclusively in Matlab; the Image Processing Toolbox was used when possible to speed development and runtime.

II. The Training Set

The training set consists of 99 images: 3 each of 33 different paintings taken at the European wing of Stanford's Cantor Center for Visual Arts. The pictures were taken with a Nokia N93 camera and are 3 Megapixels in size.

A qualitative analysis of the test images identifies some design concerns: first, all paintings are indoors under a relatively constant level of light. All light sources are fixed. There is, however, some variation in brightness likely due to the camera, flash, level of overhead lights, etc.

All images also contain a fair amount of noise; the shadowed regions are especially noisy. Some images have slight motion blurring.

The paintings of note are toward the middle of the frame; this system does not attempt to identify more than one picture per frame. The paintings may be subject to slight variances in rotation or perspective (the picture may be taken at an angle).

Many paintings have similar color schemes owing to the shared period and/or themes of the art.

Finally, the frames of all but two images have large gold regions; this fact is used when designing the painting recognition portion.

III. System design

Throughout this section the following photo of *Lament of St. Peter* from the training set is used as a visual example:

Lament of Saint Peter
Claude Vignon, 1623-1630

A. *Preprocessing*

The main function `identify_painting` accepts a 3-dimensional array of unsigned 8-bit integers (M*N*{RGB}). At three megapixels, this is 9 million elements, which makes the algorithm run over the acceptable 1-minute running time. During prepreocessing, this image is resized to 25% of its original size. (Note: 50% was originally used but 25% made the algorithm run ~3x faster; see V) Making it Fast, below.) The array is also converted from uint8's to 64-bit doubles, the native datatype of the algorithm.

B. *Painting Extraction*

This section of the system accepts an RGB image and returns a cropped and perspective-corrected image of the painting in question.

1) *Painting region identification*

This phase's task is to obtain a mask of the image where the painting's pixels are 1 and all other pixels are 0.

The cell phone photograph is converted from the Red/Green/Blue to the Hue/Saturation/Value color space. The Hue and Value channels are unused in this algorithm, but the Saturation channel represents the intensity of a color and gives a very clear signal as to the location of the frame. Because the walls are white or at most off-white, their saturation values will be low. Bright gold frames and richly-colored paintings will have high saturation values.

RGB Image

Saturation Channel

(Note: One exception to this rule is paintings with black frames which have low saturation values--"Sea View," for example; this case is handled separately below.)

The saturation channel is useful but sometimes noisy, especially in shadow regions; the noise is mitigated using Median filtering (`median2`) which was empirically determined to be slightly more useful than alternative filtering methods such as such as `wiener2`.

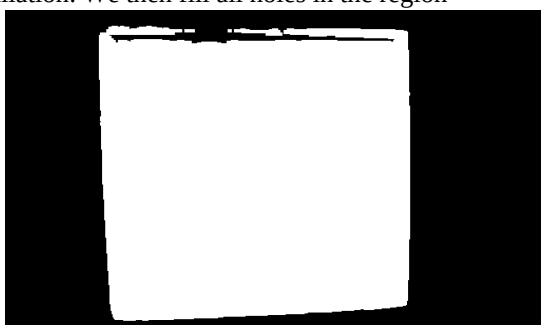
Once the noise is reduced, we threshold the saturation channel to convert the array to a binary image where 1-valued pixels represent the foreground and 0-valued pixels the background. Adaptive thresholding using Otsu's method[3] is used (Matlab function: `graythresh`). In essence, this method considers all 255 possible grayscale thresholds and attempts to find the one which most cleanly divides the histogram of intensity levels in two.

Once we have obtained the ideal thresholding level, we convert our grayscale saturation channel to a binary image of foreground/background regions.

(Special case: At this point we consider the special case of black frames: since they have low saturation values, the frames will be considered background pixels. We compute a second binary image, where 1-valued pixels are those with Red, Green, and Blue all less than 0.15 of maximum. This mask is then logical-OR'd with the grayscale mask).

Binary Image

At this point we have a mask of "foreground" pixels. There may be some imperfections; areas of reflection on the frames will show up as background pixels, for example. To alleviate this, binary dilation with a square structuring element 4 pixels in diameter is used. Internally, this is implemented as a two-pass dilation. We then fill all holes in the region



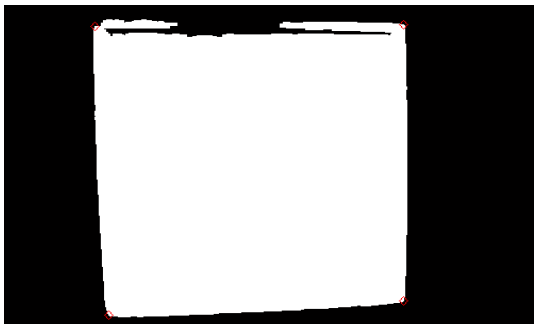
Dilated and filled image
Corners are more defined, painting is filled.

We then use region labeling to identify 8-connected regions of interest in the painting. The painting region is selected by finding the region closest to the center of the photograph, then ignoring all other regions: this approach is somewhat naïve and may come to conflict with more artistically-composed museum photographs (using e.g. the rule of thirds), but works well with respect to the given images and avoids issues such as large paintings at the borders of the photographs.

2) Perspective Correction and Cropping

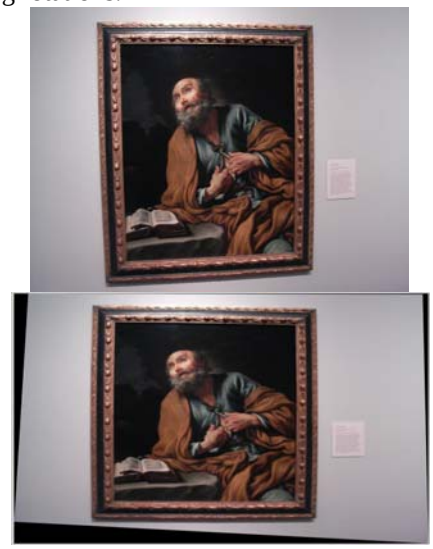
At this point in the algorithm, we have identified a region that we will consider our painting. The region, however, may not be rectangular and will often be a small subregion of the entire picture.

First, we identify the corners of the painting. Another ad-hoc algorithm is used here; we find the pixels closest to the corners of the photograph. Using Manhattan (rather than Euclidean) distance, this amounts to sweeping the line x=y across the image, finding the first intersection and calling this the top-left corner. The process is repeated to find the other three corners. More complex edge/corner detection was originally developed for this portion of the system but resulted in more errors: given that the variance in perspective and rotation is relatively controlled, this naïve approach is able to quickly find corners.



Corners identified as red diamonds

We calculate the bounding box that fits around the four corners. We then construct a perspective transformation mapping the four corners of the painting to the four corners of the bounding box (Matlab's `maketform('projective', ...)` deserves most of the credit here). This transformation is applied to perspective-correct the input image. This corrects perspective distortions very well and also does a decent job with handling rotations.



Before and After Perspective Correction

The same transformation is applied to the mask, and we find the perspective-corrected region of interest (1-valued pixels). The perspective-corrected input image is cropped to this region, and the result is a simulated head-on view of the painting in question.

The end result

C. Painting Identification

This section of the system uses the Eigenface approach described in EE368[1] to identify the input images as a certain painting. Other lecture notes and tutorials [4][5][6] were used as references in the development of this section.

The Eigenface approach is often used to match a face to a name. Pioneered by Turk/Pentland[4] and Kirby/Sirovich[5], it is a trusted algorithm for biometric face recognition. Its drawbacks include variances in lighting, angle and expression which should not be an issue for paintings: our lighting is relatively controlled, we corrected for angle in the previous stage of the system, and paintings, paradigms of unmoving objects, do not change expression.

1) Creating the training set

This algorithm requires some computation of data from the training set. The downloaded code contains the precomputed data generated from this phase of the algorithm.

Each extracted painting from the training set (size = 99 images) is taken as input, denoised, and converted to grayscale. Each input image must be the same size, so we rescale (using bicubic interpolation) to 450x450 pixels.

The 99 images are converted to column vectors of length (450x450 = 202500). They are averaged to obtain the average painting:

Averaged Painting from Training Set:
A great place to start for the aspiring artist!

This mean painting is subtracted from each input image. We concatenate all 99 column vectors in a 202500-row by 99-column matrix, **A**.

We now calculate the eigenvalues and eigenvectors of this matrix. The Sirovich and Kirby method (described in [1], also mentioned in [2]) is used at this stage of the algorithm. Finding the eigenvectors of **A** would require calculating from a 202500x202500 matrix (since the matrix must be square). We may instead calculate the eigenvectors of $\mathbf{A^TA}$, or A multiplied with its transpose. $\mathbf{A^TA}$ is 99x99; calculating the eigenvectors of this matrix is much faster (tractable for realtime applications).

Note that $\mathbf{A^TA}$ is a matrix of covariances – each element is $(A_{xy} - u_{xy})(A_{x'y'} - u_{x'y'})$.

At this point we have 99 eigenvectors and 99 corresponding eigenvalues. We multiply the training image set **A** (Matrix of size 202500*99) by the matrix of eigenvectors (99 by 99) to obtain a set of "eigenpaintings," a collection of 99 vectors of size 202500 representing mathematically significant attributes of a painting. These may be displayed as images, but are not as immediately recognizable as the ghost-like eigenfaces obtained when using thie algorithm for face identification.

Note: as described in the literature, we may discard some of the eigenpaintings which have lower-valued eigenvalues. This system does not do so only due to time constraints. It's likely we could discard many of the eigenvectors to obtain a speed boost.

We then consider each input image in the training set. Any image may be approximated with a linear combination of the given eigenpictures, so we compute the collection of 99 coefficients (essentially "how much" of each eigenpainting is in the input image). We do this by taking the dot product between each input image and each eigenpainting.

This gives us a scalar; after we do this for each eigenpainting, we have a vector of 99 coefficients (one for each eigenpainting).

In summary: mathematically, this process is a projection of the input images from "pixel space" into "painting space," a process which reduces the dimensionality from 202500 pixel components to 99 eigenpainting components. It's similar to using the Fourier Transform to represent a signal as a combination as sinusoids.

We store the collection of eigenpaintings to a data file. We also store the collection of 99 coefficient vectors (again, one for each training image).

2) Identifying paintings

At runtime, we accept an RGB image. We perform some denoising and resize the image to 450x450.

Like the training images, it is converted to a collection of coefficients representing the presence of each of the 99 eigenpaintings: the mean image is subtracted from the input, and we compute the dot product of the resulting vector with each eigenpainting.

We now have one test vector of coefficients **v**, and 99 training vectors $t_1 \ldots t_n$. To identify which of the training images is closest to this input image, we calculate the Euclidean distance (using the L2 Norm) between **v** and each $t_1 \ldots t_n$, The t_i which has the lowest distance to **v** is considered a "match."

We know the title of training image t_i, so we return it as the title of the identified painting.

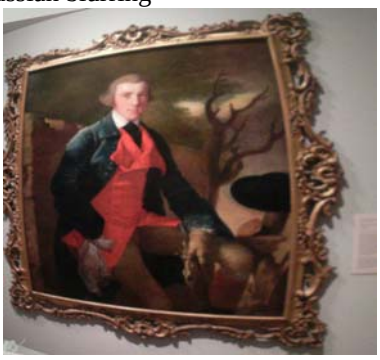
IV. Results: Accuracy

The training set consists of 99 images, 3 each of 33 paintings. To test the algorithm, it was trained with 66 of these images (two per picture) and the remaining images were fed as input. The process was repeated twice, rotating all pictures between input and training groups.

The system was able to correctly identify all paintings in these tests. The system was then trained with all 99 available images; this data is stored as data.mat and included with project submission.

A second set of tests was done to check system robustness. Training set images were hand-edited to introduce undesirable features that could result from everyday photographic situations:

- Increased noise
- Decreased lighting (uniform across image).
- Slight variations in color balance
- Rotation
- Perspective warping
- Gaussian blurring



Example modified test image

The system was still able to able to identify the images. Extreme experiments were not performed, as the test set should be close to the training set. Of note, the following assumptions were made; breaking them will likely break the system.

- Painting is contained completely within the image
- Image will be correctly-oriented (90-degree rotations are not allowed).
- Object of focus will be close to the center

V. Results: Speed (Making it Fast)

Before optimizations, the total time to identify a painting took approximately 25 seconds on a single-threaded consumer-level PC (1.5GHz Athlon, 512MB of RAM). Matlab's built-in profiler was used to identify slow regions of the code and optimize these.

Much of the time was initially spent in median filtering during the painting extraction phase. Downsizing the input by another factor of two (from 50% to 25%) dramatically sped this up. This is a speed/accuracy tradeoff, but 25% seems to be safe given the uniformity of the conditions under which the photos were taken.

A few algorithmic improvements were made, namely switching from Euclidean to Manhattan distance-from-corner for the perspective correction steps saved some time.

Using built-in Matlab functions when possible shaved more time: custom-made gray level selection was replaced by the Image Processing toolbox's `graythresh` implementation of Otsu's method.

There are some TODO's left in the code regarding other optimizations which were not implemented, for example using `bwpack` to enable a faster image dilation algorithm. Many of these were not needed, as the affected code ran in a few tenths of a second even without optimizations.

The resulting algorithm takes approximately 10 seconds to complete on the same machine.

When running on a more powerful workstation such as myth.stanford.edu, the algorithm takes less time—approximately nine seconds to identify an image. The workstation is more powerful than the development PC used, but it is also possible that the performance increase comes from multithreading: the newest version of Matlab at the time of this writing (2007a) offers support for multiple processors, and this system is heavy on very parallelizable operations (namely matrix multiplication and filtering).

On a final note, four of these nine seconds are spent loading the data file; the global/persistent commands to avoid the redundant work did not work on Linux as they did on Windows, so the entire datafile is loaded each time—I ran out of time to fix this issue at the end of the project.

A specific speed breakdown obtained on myth.stanford.edu to identify one painting:

Step Name	*Time (Seconds)*
Load 150MB dataset:	4.781
Resize input to 25%	0.844
Obtain binary mask of painting:	1.114
Perspective Correction, cropping:	1.231
Identification (eigenpaintings)	0.901

Total: **9.1 seconds**
Total (minus dataset loading): **4.32 seconds**

VI. Conclusions

The approach used to identify paintings was successful given the controlled conditions. The Eigenface-like recognition system works particularly well as we are able to avoid its pitfalls (changes in pose, variances in light, angle of image). As an alternative, a Fisherface-based approach could be considered for situations where light is not as controlled.

The system is able to correctly identify paintings taken under the same conditions as those in a training set. The speed of the algorithm shows that such technology could be applied to identify objects of interest in realtime or quickly tag objects in a photograph.

References

[1] EE368 Lecture Notes. Prof. Bernd Girod, Spring 2007

[2] M. Turk, A. Pentland, Eigenfaces for Recognition, Journal of Cognitive Neurosicence, Vol. 3, No. 1, 1991, pp. 71-86

[3] Wikipedia, Otsu's Method: http://en.wikipedia.org/wiki/Otsu's_method

[4] L. Sirovich and M. Kirby (1987). "Low-dimensional procedure for the characterization of human faces". *Journal of the Optical Society of America A* **4**: 519–524.

[5] M. Turk and A. Pentland (1991). "Face recognition using eigenfaces". *Proc. IEEE Conference on Computer Vision and Pattern Recognition*: 586–591.

[6] (Summary of Turk/Pentland referenced in my MATLAB code: used as a reference during development of identification portion of the algorithm): Summary: "Eigenfaces for Recognition. Ed Lawson http://cs.gmu.edu/~kosecka/cs803/Eigenfaces.pdf