Pictures at an Eigen-Exhibition

Christopher Tsai, June Zhang, and Ignatius I. Janatra
tsaic@stanford.edu, junez@stanford.edu, ivan87@stanford.edu
Department of Electrical Engineering, Stanford University

***Abstract*—"A picture is worth a thousand words," but finding the right thousand words to describe a picture has no simple adage. Camera phones and other mobile devices can easily capture a scene, but *processing* the image to inform its contents and describe the scene requires efficient object recognition and identification. In an effort to augment reality in a potential virtual museum guide, this document presents an efficient method to identify a painting centered in a camera phone digital image. This expedient and accurate algorithm taps the principal component transform to map the thirty-three paintings of interest into an alternative subspace spanned by an orthogonal basis of eigenimages. Within the eigenspace, Euclidean distance computation and subsequent feature recognition simplify into juxtaposition of the transformed image coordinates and the known eigencoefficients of our thirty-three known paintings.**

***Index Terms*—camera, painting, object recognition and identification, principal component analysis (PCA), eigenimage**

I. Introduction

CELLULAR phone and mobile device technology have evolved beyond mere conversational communication to support numerous forms of multimedia, the most prevalent of which is image data. Camera-phones can capture scenes and store their pictures in digital formats like the JPEG file, at a resolution sufficiently fine for detailed object recognition, whether it involves face discrimination, or, for our application, painting identification. Furthermore, by properly recognizing the features in a painting, the portable device can augment its user's reality by supplying additional information about the subject, such as the painting's title, artist, date, and background information. Nevertheless, expedient identification begins with digital image processing, and the following document examines eigenanalysis – otherwise known as principal component analysis (PCA) – as an efficient means to accurately identify works of art, using examples from the European art gallery at the Cantor Arts Center to train and test our algorithm. We characterize successful identification by our algorithm's ability to distinguish thirty-three different paintings from this European art gallery.

II. Pre-Processing Procedure

A. Downsampling

Even at its coarse resolution (compared to camera images), phone-captured image files are saturated with information. For example, the 2048 × 1536 JPEG images taken with the Nokia N93 might provide less detail than a full-fledged digital camera photo, but the amount of pixel redundancy allows us to downsample the images eightfold while preserving recognizability of imaged objects. Consider the following digital image of the painting *Edward Becher:*

Fig. 1. Original 2048 × 1536 JPEG color image of *Edward Becher*.

If we downsample the image by a factor of four, we obtain the following comparatively detailed version:

Fig. 2. Downsampled 512 × 384 color image of *Edward Becher*.

Similarly, downsampling the image by a factor of eight also reduces detail but preserves recognizability of key features:

Fig. 3. Downsampled 256 × 192 color image of *Edward Becher*.

While the naked eye can no longer discern Mr. Becher's facial

features, the shape of his hair, the colors of his jacket, the limbs of the tree, and other key objects remain distinct. The downsampled image has lost some of its fine detail, but it remains distinctly and distinguishably *Edward Becher*.

The advantages of downsampling far outweigh its minor drawbacks; while the smaller, downsampled images display less detail when compared to their full-sized counterparts, the downsampled versions require far less time to process, making them crucial to an augmented reality application in which expedient description should quickly follow a photographed scene. The number of pixels that an image processing algorithm must process decreases as the square of the downsampling factor: when we sample our original 2048 × 1536 image down to its 256 × 192 rendition, the image has 64 times fewer pixels, so any global or element-wise image processing matrix operation performs 64 times fewer computations, cutting processing time by 64 for our eightfold downsampled image, and by 256 for sixteen-fold downsampling. However, downsampling by even greater factors reaps little additional benefit in processing time:

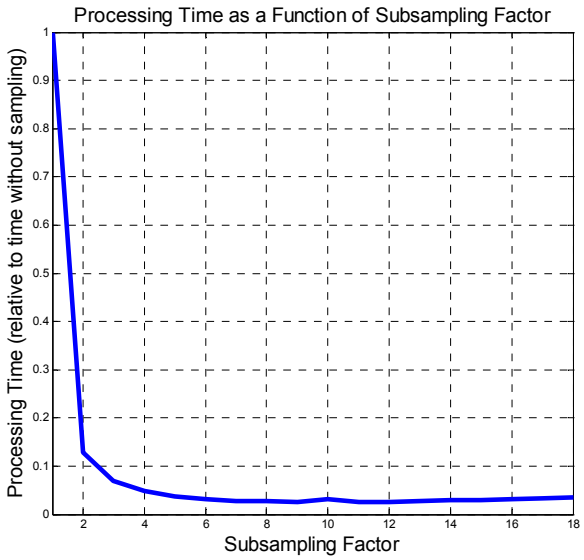

Fig. 4. Approximately inverse relationship between the speed of pre-processing and the factor by which we downsample the original image.

In the digital domain, each image is a matrix. For the human eye, the loss of detail may arguably reduce a painting's singular qualities, but, for the digital eye used to distinguish one painting from a finite number of others (such as the thirty-two other paintings in the European art gallery), the downsampling retains recognizability for the simple reason that hundreds of pixels still remain to separate one image matrix from the finite number of other possibilities.

As a matter of fact, we will shortly see that the reduction in pixel redundancy also facilitates edge detection, as bridging gaps between frame lines requires less smoothing and smaller structuring elements than would be required for the unsampled image.

B. Grayscaling

Continuing the quest for compact representation and hence minimal computation, we prune our image further by coalescing information from the three color channels into a single grayscale intensity. While color remains a powerful visual discriminant between different paintings, its limited value in principal component analysis does not warrant the threefold increase in computation that retaining all three channels would require. Thus, in an effort to reduce the amount of unnecessary information in our algorithm operand, we convert each input image from the Red-Green-Blue (RGB) color space to grayscale values between 1 and 256.

Following the transformation used in Matlab's `rgb2gray`, we merge the color channels in the linear combination:

$$f_{gray} = 0.29894 \cdot f_{red} + 0.58704 \cdot f_{green} + 0.11402 \cdot f_{blue}$$

Because the coefficients add to unity, this weighted sum preserves the intensity scale while accentuating the green channel because of its strong relative contribution to the luminance or human-perceived brightness of a pixel:

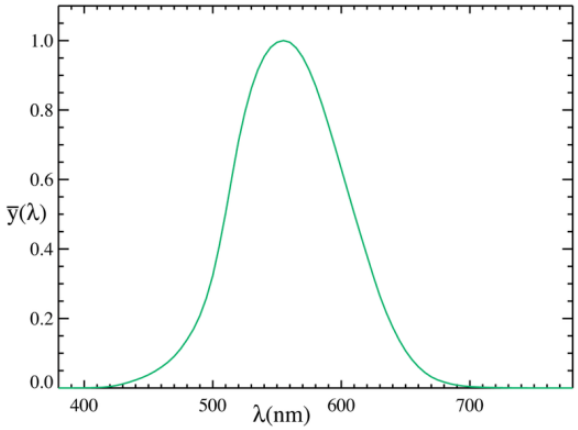

Fig. 5. Luminance efficiency curve. Maximum value occurs around a wavelength (λ) of 550 nanometers, in the middle of the green channel.

Recalling that the luminance efficiency curve peaks at green wavelengths, we ensure that the green channel contributes doubly to the grayscale intensity as the red channel, and more than five times as much as the blue channel, whose luminance contribution is minimally present in human perception.

C. Edge Detection

Prior to identifying any painting in an image, the algorithm must first locate the painting within the image space and separate it from its surroundings. To determine where a painting ends and where the wall around it begins, our algorithm detects edges by their exceptionally strong response under convolution with the bidirectional Sobel operators:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} -1 & 0 & 1 \\ 2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Filtering the image with this pair of Sobel operators extracts the horizontal and vertical image gradients, which are highest where the grayscale intensity changes most rapidly. Because the transition in brightness from a painting frame to its surrounding wall typically produces the sharpest pixel value changes in the image, the Sobel operators readily extract the lines separating each frame from the wall on which it hangs. From empirical trial and error with the data from our Nokia N93 images, we

declare a pixel in the resultant filtered image to be part of a line or object edge if its Sobel gradient exceeds 0.052 on the unit intensity scale [0, 1]. For example, consider the following downsampled snapshot of *Napoleon*:

Fig. 6. Downsampled grayscale image of *Napoleon*.

Filtering this image with subsequent passes of the Sobel operators with a threshold at 0.052, we obtain the edge map:

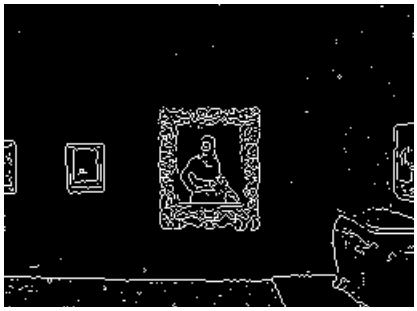

Fig. 7. Sobel gradient edge map of *Napoleon*. Not all pixels connect.

As desired, the painting frames yield the largest Sobel gradient values, so their edges protrude most prominently in the gradient output. However, because these edge points may not form a closed boundary or even an unbroken line, we bridge the gaps in the edge map by morphologically closing the image with a small isotropic structuring element, such as the radius-one diamond:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Morphologically closing the binary edge map not only connects fragmented frame edges but also smears false edges together, ensuring that the resulting image contains closed loops – some of them false ones – rather than single isolated points:

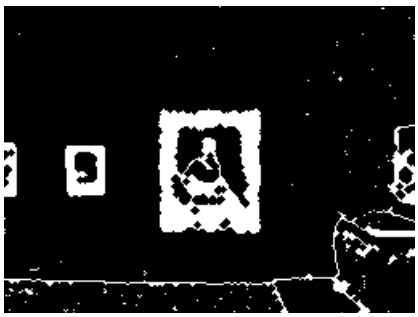

Fig. 8. Morphological closing of the Sobel edge map with a radius-one diamond structuring element. Lines are bridged, but not all holes are closed.

Finally, noticing that many closed edge loops still contain holes, we fill the image by setting all values within each boundary to unity using an algorithm like Matlab's `imfill`:

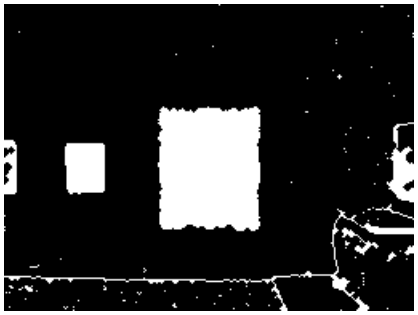

Fig. 9. Filled morphological closing of the Sobel gradient.

At this point, our processed image contains a sufficient number of well-defined areas that we can proceed to isolating the region of interest. However, before we proceed, we return again to sampling.

D. Edge Effects of Image Downsampling

As we previously discussed, sampling reduces the size of the input image. Smaller images require less processing time.

More importantly, downsampling removes many of the small noise-induced fake edges in the original image. Decimation also sharpens the desired frame edges, facilitating gap bridging by removing the need for larger structuring elements; the simple radius-one diamond easily bridges most downsampled image edges. In conclusion, the downsampled image is more robust for frame extraction. The following figure illustrates the improvement in edge detection gained merely by eightfold decimation. As shown below, morphological closing becomes more effective after downsampling the image:

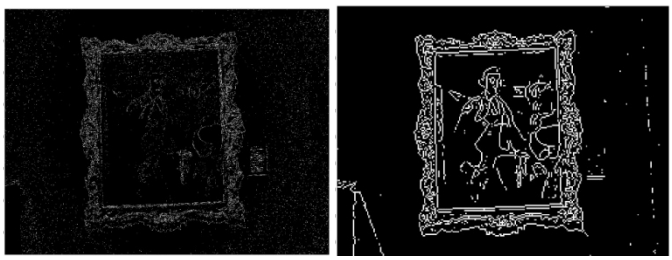

Fig. 10. Sobel edge map of *Edward Becher* when operating on (left) the original image, and (right) on the eightfold downsampled image. With fewer pixels to bridge, the decimated image responds better to closing.

E. Region of Interest Isolation

An image typically comprises several secondary objects in addition to the primary painting of interest; for example, in addition to *Napoleon*, three other paintings, three labels, and a wooden dresser appear in the image, all of which could potentially confuse our image processing algorithm when it attempts to identify the central painting. Assuming that the camera user points her device at the object of interest, we need to identify only the central painting, so we must eliminate the regions in which the central pixel does not belong.

To check whether or not a region contains the central pixel in the image, we would need to both compute each region's bounding box and repeatedly evaluate the Boolean Matlab

function `inpolygon` on bounding box vertices. We might also consider checking the region itself rather than its bounding box for the central pixel. However, if `imfill` failed to completely fill a region because of concentric boundaries or improperly closed loops, then the central pixel may not belong to *any* region, rendering even exhaustive search futile in a debacle such as the one pictured below:

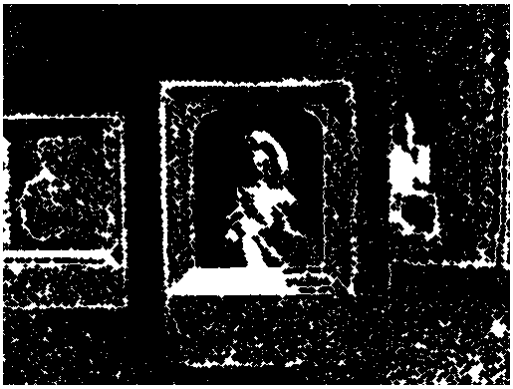

Fig. 11. Example of failed (incomplete) filling of the morphologically closed edge map. Because closing failed to bridge peripheral gaps, the central region of the painting *Adam and Eve* does not necessarily contain the center pixel.

Fortunately, even if morphological closing fails to close boundary loops, and even if the interior of the painting frame is not entirely unity following `imfill`, the region of interest will still boast one of the highest pixel counts out of the perimeter's sheer size. Hence, instead of checking the containment of pixels in each region, our algorithm instead labels regions using `bwlabel` and ranks all 300-500 regions by their unity-valued pixel count, which we can calculate either by looping or using

```
regionprops(filledImage,'Area')
```

thereby retaining the dominance of the painting regions even when they are improperly closed or incompletely filled. Having narrowed down the possible regions of interest to large clusters, we can afford to compute all candidate bounding boxes to ascertain containment of the central pixel. We select the region with greatest area if its bounding box contains the central pixel, and opt for the second-largest region otherwise, assuming that an image rarely features a single small painting surrounded by two larger ones. Finally, we still must tap `inpolygon` to ensure that we choose the central painting, but proper selection no longer relies on flawless filling or exhaustive search across all bounding boxes, cutting the number of `inpolygon` calls from hundreds to one or two. In our example, the largest region is our answer:

Fig. 12. Largest region of the *Napoleon* image isolated from surroundings.

Before we proceed with region-based processing, we note here that our image now informs whether or not our Sobel gradient edge map threshold successfully isolated the frame.

If motion blur or alternative frame design somehow causes a frame edge to transition more smoothly into its surrounding wall, then edge detection with a high threshold of 0.052 may fail, so our algorithm must then lower the threshold and retry Sobel edge detection.

Initially, the program has no way of knowing whether or not this empirical threshold unambiguously locates the frame edges, but we can foresee failure after filling regions from the edges. Certain measures that indicate the failure of our initial threshold include wild elongation of improperly encircled regions, or simply a lack of clustered pixels after the fill operation. Once we obtain the central region of interest, we check it for elongation and irregular clustering.

If Sobel segmentation fails to highlight a significant portion of the frame, then pixels within the artwork may connect to form a region comprising only part of the painting, as shown below:

Fig. 13. Example of failed region segmentation due to over-thresholding. The triangular shape (and therefore proportion of the bounding box) of the resulting region raises the flag, leading to a lowered threshold in the next trial.

In this case, the segmented region of interest may be either extremely elongated or strangely shaped. For example, if the aspect ratio of the region's bounding box exceeds 2.4 or dips below 0.55, then segmentation likely cropped a significant portion of the painting, since no painting exhibits such elongation. Furthermore, if the number of pixels in the region amounts to less than 7/10 of its bounding box area, then the segmented region must not be rectangular, again indicating failed frame edge detection. If either of these scenarios unfolds, then we backtrack to Sobel gradient filtering with an empirically lowered threshold of 0.042. Following subsequent region isolation, we again perform the same two checks, lowering threshold further if necessary. We thus decrement the Sobel threshold every time the central region exhibits extreme aspect ratio or unusual (non-rectangular) shape, stopping only after we obtain an approximately rectangular central region.

After isolating the largest region, our algorithm smoothens the region boundaries by computing the convex hull; because the convex hull is the smallest convex polygon that can contain the region, it subdues pointed edges or jagged boundaries, tempering sudden changes with longer, sweeping sides. Our algorithm exploits the Matlab command:

```
regionprops(regionImage,'ConvexImage')
```

Fitting the convex hull not only smoothens region boundaries but also crops the image to its bounding box, thereby eliminating all background pixels.

F. Corner Detection

Because of perspective-based tilt and oblique camera angles, the painting on the wall does not always appear rectangular in the image plane, such as the *Daughters of Jethro* image below:

Fig. 14. Image of *Daughters of Jethro*, distorted by camera angle.

Our algorithm must recognize paintings whether or not they directly face the camera, but the principal component transform cannot possibly transform two differently angled paintings into the same point. To standardize all paintings of interest before applying the principal component transform, our algorithm must rectify the perspective distortion of parallelogram-shaped regions with an inverse perspective transform. Matlab supplies a control-point-based perspective transform but requires specification of a region's four corners.

Numerous sophisticated algorithms for corner detection such as the Haralick and Harris corner detectors already exist. However, these advanced algorithms supply superfluous detail in their corner detection:

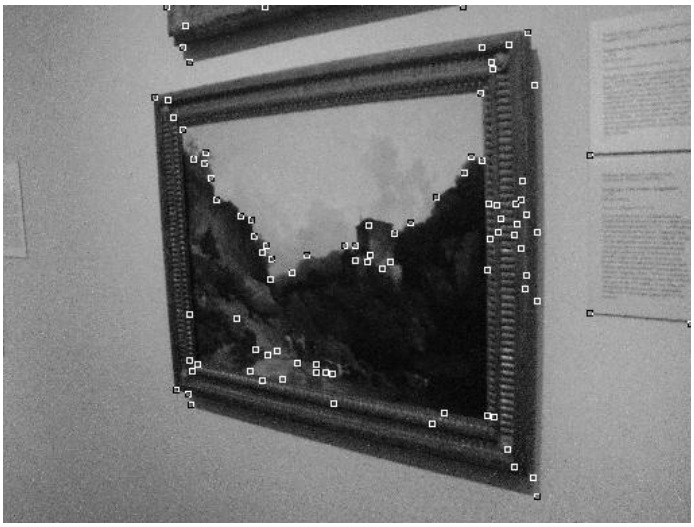

Fig. 15. Excessive detail in Haralick corner detection.

Because the Haralick corner detector spots even slight intermediate corners along the painting sides, we opt for a simpler method.

As control point transform requires only four corners, we divide our convex region into four quadrants and locate the point in the region with the maximal displacement from the image center:

$$\max_{\text{convex hull}} [|x - x_0| + |y - y_0|]$$

where the region origin is (x_0, y_0). The maximum in each of the four quadrants constitutes the four corners of our region. For instance, suppose we input the convex hull of the tilted *Daughters of Jethro*:

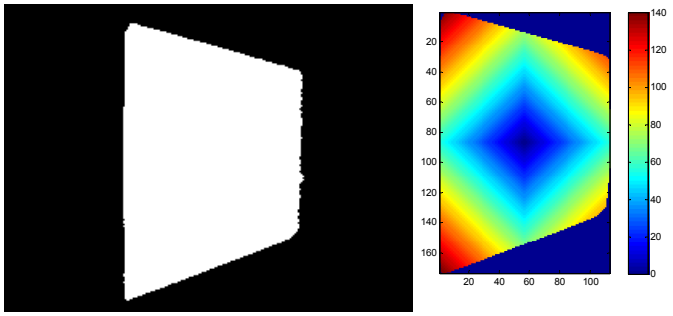

Fig. 16. Sum of coordinates as a measure of the degree of cornerness.

The red regions display the largest sum $[|x - x_0| + |y - y_0|]$ in the convex hull, indicating the highest degree of cornerness in the painting. By locating the critical point in each of the four quadrants, we deduce the locations of the four corners.

After repeated trial and error, we remark that the corners always occur along the region boundary, so we can expedite corner detection by maximizing along the *perimeter* of the convex hull instead of its entirety; we then need search only a sparse matrix, thereby reducing the number of required computations. Juxtaposing the two matrices, we see that the perimeter corner detection yields the same result despite operating on the incomplete region:

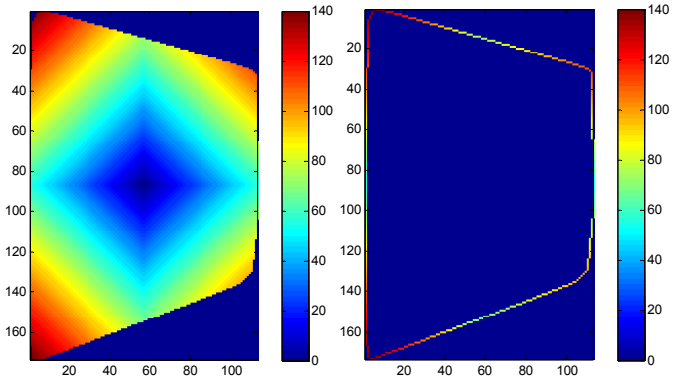

Fig. 17. All four corners occur along the perimeter, so corner detection need not examine the interior of a region. Sparse matrices expedite computation.

With the corners of the convex hull located, our algorithm proceeds to invert the perspective transform.

G. *The Inverse Perspective Transform*

Matlab's `cp2tform` command requires two sets of control points – one for the distorted image and another for its rectangular alignment. The four corners previously detected provide the perspective-distorted input points, but we can also use the same corners to estimate the base points:

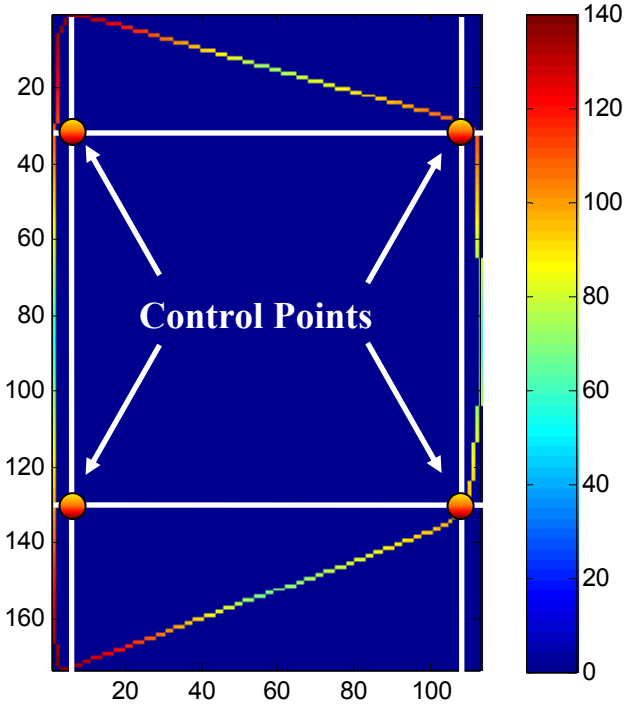

Fig. 18. Intermediate x and y coordinates define a rectangle to which an inverse perspective transform can map the distorted quadrilateral. Intermediate values come from the sorted corners.

As the graphic above reveals, the two intermediate *x*-coordinates and two intermediate *y*-coordinates define a box that approximates the painting's rectangular shape. In this manner, the four control points share coordinates with the sorted corners previously obtained; having previously computed the distorted painting's four corners, we need only order them and extract the second-highest and second-lowest *x* and *y* corner coordinates to form the rectangle shown above. This method provides a workable approximation only when the camera angle from the wall is not too severe; there is no hope for recovery from extreme degrees of perspective distortion.

With these control coordinates, Matlab generates an inverse projective transform with the command:

```
cp2tform(cornersXY, rectXY, 'projective')
```

where `rectXY` lists the four intermediate pairs shown above. Upon performing the previously generated inverse perspective transform using `imtransform`, we obtain the image:

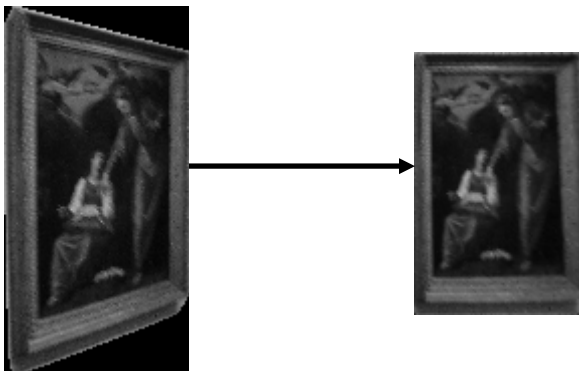

Fig. 19. Inverse perspective transform rectifies camera angle distortion.

Although the transformed *Education of the Virgin* closely resembles the frontal view of the original, eigenanalysis requires standardized dimensions, so, in order to align all of our images on the same dimensional scale prior to principal component decomposition, we perform one additional affine transform to square the rectangular image to 128 pixels × 128 pixels:

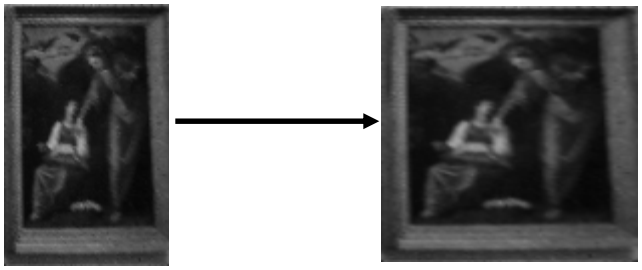

Fig. 20. Affine transform maps rectangular painting to a 128 × 128 square image, so eigenanalysis can operate on images of the same dimensions.

III. Eigenprojection

Thirty-three paintings exude obvious visual differences in Cartesian space, but the variation in brightness and shadow from one digital image to another complicates automatic identification, since the slightest flash spot or differing camera angle captures an entirely different image even when the subject remains the same. Ultimately, the matrix of pixel values that results from a digital image depends more heavily on the lighting conditions and camera geometry than on the finely nuanced painting pigments themselves. Considering the sensitivity of matching in Cartesian image space, we base our identification in a different space – an eigenspace spanned by orthonormal basis images, otherwise known as *principal components* or *eigenimages* **[2]**.

Given thirty-three paintings of interest, we stack their registered square matrices, one on top of the other, to form a 128 × 128 × 33 rectangular prism of images:

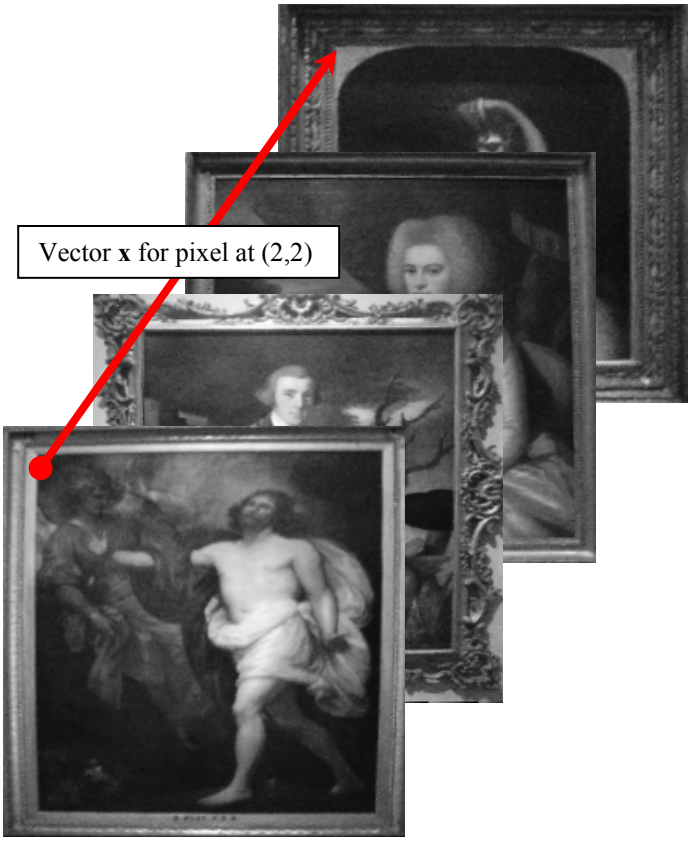

Fig. 21. Upon stacking 33 training images (not all pictured), we define the vector population X comprising the vectors through each set of pixels.

Each point in the 128 × 128 grid defines a 33-element vector. Moving through the stack, this vector contains one pixel from each of the 33 images, allowing us to describe the image stack by a population of such $\mathbf{x}$ vectors, one for each point in the 128 × 128 grid. Arranging these 16,384 vectors as columns of a matrix X, we can describe the image set by a mean vector $\boldsymbol{\mu}_x$ and a covariance matrix $\boldsymbol{\Sigma}_x$, defined to be

$$\boldsymbol{\mu}_x = \frac{1}{128 \times 128} \sum_{k=1}^{128 \times 128} \mathbf{x}_k$$

$$\boldsymbol{\Sigma}_x = \frac{1}{128 \times 128} \sum_{k=1}^{128 \times 128} \left[\mathbf{x}_k \mathbf{x}_k^T - \boldsymbol{\mu}_x \boldsymbol{\mu}_x^T\right]$$

Because the covariance matrix $\boldsymbol{\Sigma}_x$ is *real* and *symmetric*, the Spectral Theorem holds that it must be *orthogonally diagonalizable* with real eigenvalues and orthonormal eigenvectors **[9]**. Thus, we can use the eigenvectors of $\boldsymbol{\Sigma}_x$ as an orthonormal basis to span the image space. However, in Cartesian (image) space, the vectors bear strong correlation, as $\boldsymbol{\Sigma}_x$ is not a diagonal matrix; in fact, the off-diagonal entries can be quite large, indicating that the vector of pixels through one point in the image grid does not differ considerably from its neighboring vectors. Such a vector comparison in the image space will still yield viable identification, provided that our algorithm accounts for the nonzero correlation between pixels. However, discrimination grows much more robust when the eigenbasis is orthonormal, the correlation matrix is purely diagonal, and the main coordinate axes are spaced as far apart (90°) as possible. Hence, in the first step of principal component decomposition, we orthonormally diagonalize the image covariance matrix with an orthogonal transformation given by matrix A:

$$A\Sigma_x = \Sigma_y A$$

$$\Sigma_x = A^{-1}\Sigma_y A$$

If we choose an orthogonal transformation such as the *Hotelling transform* (widely used in PCA), then $A^{-1} = A^T$ since A is a unitary, real, and therefore orthogonal matrix **[2]**.

$$\Sigma_x = A^T\Sigma_y A$$

$$\Sigma_y = A\Sigma_x A^T$$

The orthogonal transformation matrix A orthonormally diagonalizes the image covariance matrix. As a result, Σ_y – the covariance matrix in the new eigenbasis given by the columns of A – is a diagonal matrix comprising the eigenvalues of our original covariance matrix Σ_x:

$$\Sigma_y = \begin{bmatrix} \lambda_{x,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{x,33} \end{bmatrix}$$

Most importantly, the off-diagonal elements of the covariance matrix are now zero, meaning that diagonalization has decorrelated the image vectors. The choice of an orthogonal transformation – the Hotelling transform – further provides an orthonormal eigenbasis particularly suitable for representation and identification. As we know from elementary linear algebra, the column vectors of the transformation matrix A are precisely the 33 orthonormal eigenvectors of Σ_x.

In order to construct this matrix A that our algorithm needs to perform the Hotelling transform on given input images, we first generate a model covariance matrix of 33 perfectly aligned images, as an ideal case for the 33 possible paintings that we must identify. (In actuality, we can compute the covariance matrix from any 33 training images of the 33 paintings, but, assuming that our pre-processing alignment functions flawlessly, we begin with a model set.) After computing the covariance matrix Σ_x of this model vector population X (with vectors $\mathbf{x}$ arranged as columns in 33 × 16,384 matrix X), we arrange its eigenvalues in descending order along the main diagonal of our model diagonal matrix Σ_y . We then stack their corresponding eigenvectors in the same order to form the orthogonal transformation matrix A.

Brandishing the principal components transform given by matrix multiplication with A, we are prepared to transform any training set of $L = 33$ model paintings using the algorithm **[2]**:

$$\mathbf{y} = A(\mathbf{x} - \boldsymbol{\mu}_x)$$

Thus, multiplication by the orthogonal matrix A maps vectors in the training set space of $\mathbf{x}$ into the eigenspace vector $\mathbf{y}$, in which all principal components of the thirty-three model images have been decorrelated and orthogonalized. Brightness values, grayscale intensities, and shadow no longer dominate image differentiation in this alternate subspace, since the eigenaxes have been orthonormally aligned with the precise paintings, making *direction* the primary discriminant rather than *magnitude*.

Recall the definition of a vector population X, for an image stack of N × N × L images:

$$X = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{x_1} & \mathbf{x_2} & \cdots & \mathbf{x_{N^2}} \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \in \mathbb{R}^{(L \times N^2)}$$

In image space, each of these vectors simply informs the grayscale intensity of a pixel, but we can decorrelate these intensities by transforming each image space vector into eigenspace through premultiplication by the orthogonal principal component transform matrix A:

$$A = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{v_1} & \mathbf{v_2} & \cdots & \mathbf{v_L} \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \in \mathbb{R}^{(L \times L)}$$

…where the vectors $\mathbf{v_i}$ represent the L = 33 eigenvectors of the correlation matrix Σ_x. Notice that, while the *columns* $\mathbf{v_j}$ of matrix A comprise the *L* orthonormal eigenvectors of Σ_x, the *rows* $\boldsymbol{u_i}$ of matrix A reveal the eigencoordinates of the *L* training images in matrix X, used to construct A:

$$A = \begin{bmatrix} \leftarrow & \boldsymbol{u_1} & \rightarrow \\ \leftarrow & \boldsymbol{u_2} & \rightarrow \\ & \vdots & \\ \leftarrow & \boldsymbol{u_L} & \rightarrow \end{bmatrix} \in \mathbb{R}^{(L \times L)}$$

Assuming that our images have previously been normalized to zero mean (prior to concatenation in X), transforming our image space into eigenspace entails a simple Cayley product:

$$Y = AX = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ A\mathbf{x_1} & A\mathbf{x_2} & \cdots & A\mathbf{x_{N^2}} \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \in \mathbb{R}^{(L \times N^2)}$$

Because eigenspace matrix Y shares the same dimensions as image stack matrix X $(L \times N^2)$, and since $L \ll N^2$, matrix Y is a *fat* matrix. As all training images in image stack X differ, matrices X and Y are also *full rank*.

To determine the coordinates of the given set of training images in the eigenspace, we need only recall that we constructed this very eigenbasis – the columns of matrix A – with models of the training set, so the coordinates of that training set in the eigenspace are the rows of A – a linear combination of the basis columns. The model images we use to construct the transformation A will enjoy simple representation in the eigenspace, since the orthogonal axes, by design, align with our model images.

However, our original formation of A assumed ideal model images, perfectly aligned and registered. Such perfect alignment rarely occurs in reality, as the empirical nature of edge detection and projective transformation inevitably yields variable pre-processing across images taken at different angles with varying amounts of motion blur. Thus, to ensure that the transformation is as robust as possible for all images of the same painting, we construct A with all possible variations that could arise from our training set; in addition to the 33 model images previously discussed, we insert heavily shadowed, slightly tilted, and even poorly-lit versions of an image to account for scenarios in which edge detection and/or inverse perspective transforms yield imperfect registration. Hence, even when the variability of pre-processing rears its head, our expanded eigenbasis still contains an eigenimage (or eigenaxis) that is relatively close to the imperfectly aligned input, making detection by direction as easy as if the input image had been perfectly aligned. Consequently, in anticipation of numerous forms of misregistration, we expand the eigenbasis to 57 eigenimages for the eightfold downsampling (and 66 for sixteen-fold downsampling). We choose the additions empirically to cover all scenarios that arise frequently in pre-processing, such as excessive shadow detection under *Adam and Eve* and *Portrait of a Young Artist*:

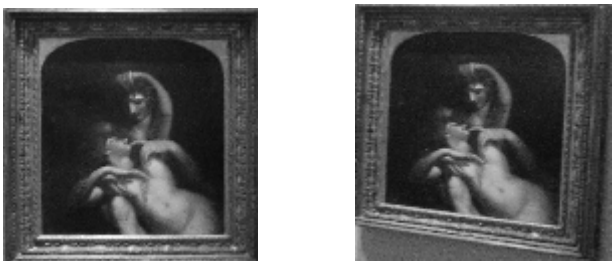

Fig. 22. Addition of a second eigenimage for *Adam and Eve* to cover pre-processing that fails to segment the painting completely from shadow.

Fig. 23. Addition of a second eigenimage for *Portrait of a Young Artist* to cover the large number of pre-processed training images with lower shadow.

In this manner, we incorporate our knowledge of pre-processing weaknesses into the eigenbasis expansion to improve the robustness of identification.

Once the images from which we generate our eigenbasis adequately represent the spectrum of possible variations in pre-processing output, we form the eigenbasis as described previously for the model set: (1.) stack the chosen set of representative images; (2.) form matrix X from the vectorized image stack; (3.) compute the covariance matrix Σ_x; (4.) calculate the eigenvalues and corresponding eigenvectors of the covariance matrix Σ_x; (5.) arrange the eigenvectors as columns of a matrix A; and (6.) transform the image set into eigenspace coordinates through the multiplication Y = AX.

With a solid representative vector population and its

eigenspace coordinates documented in Y, we are now prepared to identify any painting that aligns similarly to paintings represented in the columns of matrix X.

Suppose we receive an input image that our pre-processing algorithm squares and aligns to a size of N × N. Transforming a single painting into the eigenspace spanned by columns of A cannot be accomplished by simple matrix multiplication, since the eigenbasis involved a *stack* of 57 (or 66) images, with each pixel vector **x** penetrating the entire training set from the first image to the last, through a single point in the grid:

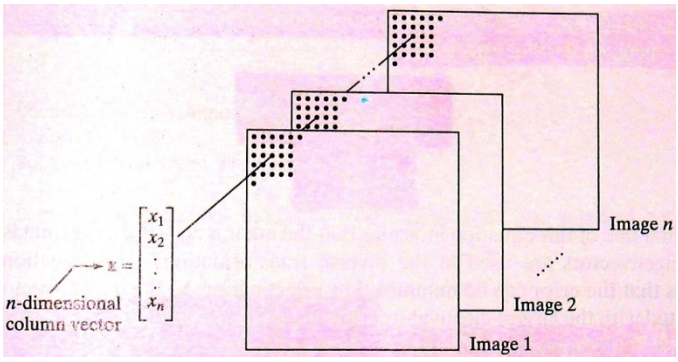

Fig. 24. Illustration of the image stack and the vectors that represent *not* the vectorized images but rather the common pixels in the stacked images. Each vector **x** does not represent a single image but rather than intensities at one location in *all* training images. **[3]**

One might wonder how a *single* image can enter the eigenspace. Consider the *transpose* of the following equation:

$$\mathrm{Y} = \mathrm{AX}$$

$$\mathrm{Y^T} = \mathrm{X^T A^T}$$

Since matrix Y is *fat* and *full rank*, its transpose $\mathrm{Y^T}$ is *skinny* and *full rank*. Consequently, $\mathrm{Y^T}$ has a *left inverse*, which we denote the pseudoinverse[i] $(\mathrm{Y^T})^\dagger$. Premultiplying both sides of the equation by this left inverse of matrix $\mathrm{Y^T}$, we obtain the equation:

$$\mathrm{I} = (\mathrm{Y^T})^\dagger\, \mathrm{X^T A^T}$$

Because the Hotelling transform is an orthogonal transformation, its matrix is orthogonal: $A^{-1} = \mathrm{A^T}$.

$$\mathrm{I} = (\mathrm{Y^T})^\dagger\, \mathrm{X^T} A^{-1}$$

$$A = (\mathrm{Y^T})^\dagger\, \mathrm{X^T}$$

In this manipulated version of the transformation equation, the transformation matrix is now $(\mathrm{Y^T})^\dagger$. We denote this matrix the *eigenprojector*. The matrix $\mathrm{X^T}$ still derives from the stacked image pixels, but its dimensions are $(\mathrm{N^2} \times \mathrm{L})$, where N is the side length of our pre-processed square image, and L is the number of images used to construct the eigenbasis; in the eightfold downsampling, our algorithm squares images to 128 × 128, so N = 128, while L = 57 images represent possible pre-processing outputs robustly. However, if we downsample further (by sixteen), we square images to 64 × 64, so N = 64. Because identification accuracy also decreases with the image size, we must add more possible outcomes in the formation of the eigenaxes, so L = 66 for the sixteen-fold downsampled case.

Nevertheless, regardless of the size of our registered square input image, its dimension $(\mathrm{N^2} \times \mathrm{L})$ means that input images *f* of dimension N × N can be pre-multiplied in the same manner by the eigenprojector $(\mathrm{Y^T})^\dagger$. The number of image stack vectors is the same size as the length of the vectorized image, so we can perform eigenspace conversion without a stack by exploiting the transpose for its identical size. The advantage is evident. Having already computed the eigenprojector offline (prior to identification), the conversion $(\mathrm{Y^T})^\dagger \mathbf{vec}(f)$[ii] exacts only one matrix product:

$$\begin{array}{rcl} g & = & (\mathrm{Y^T})^\dagger \quad \mathbf{vec}(f) \\ (\mathrm{L} \times 1) & = & (\mathrm{L} \times \mathrm{N^2})\ (\mathrm{N^2} \times 1) \end{array}$$

To which eigenaxis of A does the resultant set of coefficients $\mathbf{g}$ correspond? Recall that the *L* columns of matrix $\mathrm{X^T}$, in place of which we substituted input image *f*, correspond to the *L* images from which we constructed our eigenbasis. The resultant set of eigencoefficients $\mathbf{g}$ is therefore comparable with the eigencoordinates of our original training set, which we recall to be the *rows* of matrix A. Thus, locating the most similar painting to *f* in the given set of training images used to construct A amounts to comparing the coordinates in eigenspace; the most similar painting – ideally the painting with the same name as the painting in the input image – will point preferentially along one of the discreetly chosen eigenaxes, so its coordinates in the rows of A will closely match the eigencoefficients $\mathbf{g}$. The algorithm concludes that the input image *f* features the painting in original training image p_k if the eigencoefficients $\mathbf{g} = (\mathrm{Y^T})^\dagger \mathbf{vec}(f)$ are closest in the mean-square sense to $\boldsymbol{u}_\mathbf{k}$, the k[th] row of A:

$$\left|(\mathrm{Y^T})^\dagger \mathbf{vec}(f) - \boldsymbol{u}_\boldsymbol{k}\right|^2 = \min_{\{j=1,\cdots,L\}} \left|(\mathrm{Y^T})^\dagger \mathbf{vec}(f) - \boldsymbol{u}_\boldsymbol{j}\right|^2$$
$$image\ number = k:\ |\mathbf{g} - \boldsymbol{u}_\boldsymbol{k}|^2 \le \left|\mathbf{g} - \boldsymbol{u}_\boldsymbol{j}\right|^2 \forall\, j \in \{1, \cdots, L\}$$

Much like mean-square comparison in the Cartesian image domain, the mathematical computation required for such a computation involves only *L* comparisons, but the separation – and hence distinguishability – of the *L* images increases in eigenspace because of the deliberate orthogonality of the eigenimages. Meanwhile, by forming our eigenimages from not only the thirty-three paintings but also their pre-processed variations, each possible output has a closely aligned eigenaxis, allowing the coordinates in $\boldsymbol{u}_\boldsymbol{i}$ to be as distinct as possible.

To ensure that we correctly map the *L* possible eigenaxes to their respective paintings, we create an index vector to relate each row of A to its corresponding image space painting. Thus, once our algorithm determines the image number *k*, we can extract the painting number and title by indexing its *k*[th] entry.

[i] By definition, the *pseudoinverse* of a matrix M is defined in terms of its singular value decomposition: If $\mathrm{M} = U\Sigma\mathrm{V^T}$, then $M^\dagger = V\Sigma^{-1}\mathrm{U^T}$ **[10]**.

[ii] **vec**(*f*) concatenates the columns of matrix *f*, one below the other. If *f* is an M × N matrix, then **vec**(*f*) is an MN × 1 column vector. In Matlab, **vec**(*f*) is denoted `f(:)`.

V. Experiments in Eigenspace with Sirovich & Kirby

In addition to the aforementioned eigenaddition, we also attempted solution with the more conventional Sirovich and Kirby eigenimage decomposition. As the algorithm needs to identify only 33 different paintings, it needs only 33 eigenimages under perfect pre-processing.

As with the previous method's model set, the eigenimage solution compiles thirty-three unique, focused images, each represented by a column vector $\{\vec{\Gamma}_1, \vec{\Gamma}_2, \cdots, \vec{\Gamma}_{33}\}$ to create the training set matrix $S = \left[(\vec{\Gamma}_1 - \vec{\mu}) \quad (\vec{\Gamma}_2 - \vec{\mu}) \quad \cdots \quad (\vec{\Gamma}_{33} - \vec{\mu})\right]$, where $\vec{\mu}$ is the mean of all the vectorized images. Tapping the Sirovich & Kirby method to compute eigenimages,

$$SS^H S\vec{v_i} = \lambda_i S\vec{v_i}$$

where $\vec{v_i}$ are the eigenvectors of $S^H S$. Consequently, $S\vec{v_i}$ are the eigenvectors of SS^H, also known as *eigenimages*.

In order to reduce calculation time, the algorithm consults only the twelve eigenimages with the largest eigenvalues, three of which appear below:

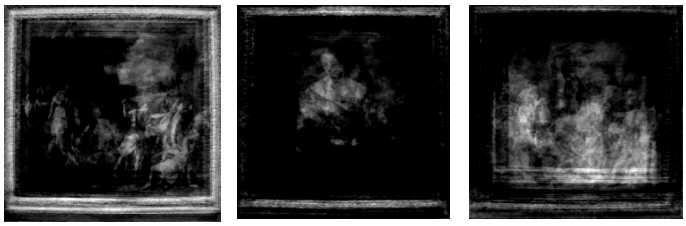

Fig. 25. Three highest-energy eigenimages derived from Sirovich & Kirby.

This algorithm maps each of the thirty-three images into the twelve-dimensional eigenimage space by finding the correlation between each image and the eigenimages. The eigenimage matrix and correlation matrix are stored in memory as

$$\Sigma_i = \begin{bmatrix} \mathrm{E}\left[\vec{\Gamma}_\mathrm{i} \cdot S\vec{v_1}\right] \\ \mathrm{E}\left[\vec{\Gamma}_\mathrm{i} \cdot S\vec{v_2}\right] \\ \vdots \\ \mathrm{E}\left[\vec{\Gamma}_\mathrm{i} \cdot S\vec{v_{12}}\right] \end{bmatrix}$$

After loading the eigenimage and correlation matrix, this algorithm calculates the correlation between the unknown image and the eigenimages, juxtaposing the mean square difference between these correlation values and the correlation values of the known 33 paintings. The most similar painting yields the minimum mean square error.

Because of our pre-processing algorithm's occasionally imperfect performance, we again introduce additional images to the training set matrix *S* in order to correctly identify blurrier and more severely tilted images.

Eigenimage identification process proves to be an extremely fast algorithm as the calculation-intensive step (e.g. calculating the eigenimages) is done prior to recognition task. In addition, eigenimage identification allows us to downsample the images to further reduce processing time. The downside of eigenimage recognition is its reliance on the consistency of pre-processing. It is also somewhat unpredictable, as expanding the eigenset does not always improve accuracy; one must constantly balance interpolation – the addition of more eigenimages – and extrapolation outside the training set. The danger of *overmodeling* always looms one poorly chosen eigenimage away.

VI. Experiments in Cartesian Space

Minimization of Cartesian mean square error (MSE) remains one of the most intuitive ways to identify similarity between images. Finding the lowest MSE is straightforward in implementation, fast in execution, and relatively robust. Over a small region, the pixel values of the processed images remain fairly constant. For that reason, the method is invariant to minor shifts incurred during pre-processing.

The algorithm taps at least one processed image of each picture as reference, while normalizing all reference images to a mean of 128 and a standard deviation of 80 to reduce the effect of uneven lighting conditions. The processed input image – converted to grayscale, centered and projected to a pre-determined size – is normalized to have the same mean and deviation as the reference images. For every reference image, the algorithm computes the MSE between the processed input and the reference. The reference image yielding the lowest MSE corresponds to the desired painting.

To properly identify the given training set of 33 × 3 images, this approach can achieve perfect accuracy with 33 reference images (one reference image per painting). However, when this set of reference images is used to identify 601 images of the expanded training set, accuracy declines to 97%. In order to achieve perfect accuracy, the reference set has to be expanded by taking more than one reference image per painting if necessary. This implementation uses 48 reference images to achieve perfect accuracy in the expanded training set.

Another equally important barometer of success is robustness to noise, blur, angle, and shadow, which we quantify with *tolerance*:

$$\text{Tolerance} \triangleq \frac{\min\limits_{all\ MSE} |MSE\ of\ incorrect\ choice|}{|MSE\ of\ correct\ choice|}$$

High tolerance signifies clarity and ease in decision-making. In these cases, minor changes in the processed input image very likely will not affect the decision. On the other hand, low tolerance signifies ambiguity. In this case, it is very likely that the decision changes direction if the processed input image is distorted by a very minor shift or additive noise.

Figure 26 displays the histogram of tolerance when identifying 601 images of the expanded training set using the set of 33 reference images, whereas Figure 27 shows the histogram of tolerance using the original set of 48 reference images:

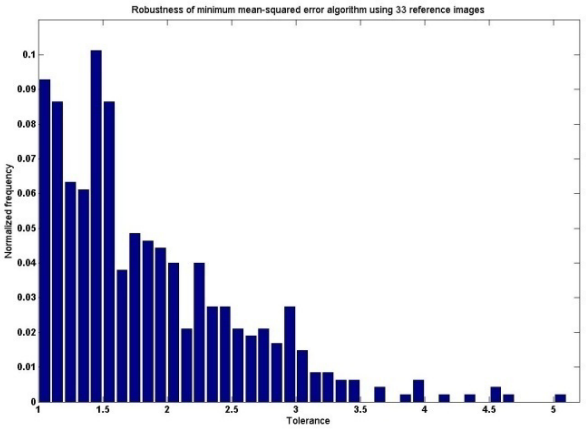

Fig. 26. Tolerance distribution as a measure of identification accuracy when making mean square difference comparison over 33 reference images.

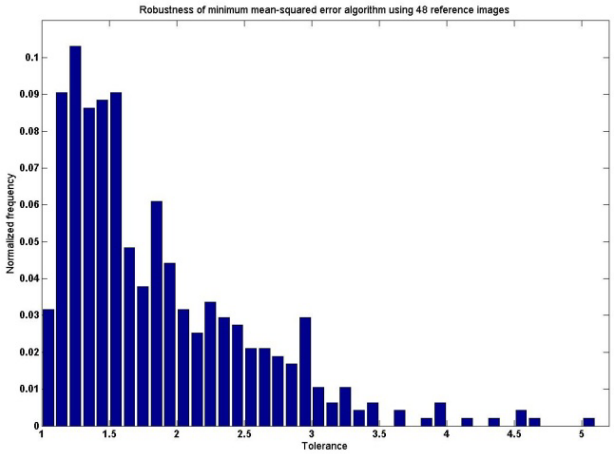

Fig. 27. Tolerance distribution as a measure of identification accuracy when making mean square difference comparison over 48 reference images.

Observe that, when we consult a reference set of 33 images, 9% of the decisions suffer from low tolerance of less than 1.1, whereas, tapping a reference set of 48 images, only 3% of the decisions exhibit tolerance less than 1.1. In conclusion, minimizing the Cartesian MSE while appropriately expanding the reference set improves not only accuracy but also robustness.

VII. Comparison

Method	10-Image Test Set Time	Mean Time per Painting	Training Set (3 × 33) Accuracy	Total Set (601) Accuracy
Eigenprojection (Downsample × 4)	21.411 sec	1.772 sec	99/99 100%	595/601 99.002%
Eigenprojection (Downsample × 8)	4.202 sec	0.1226 sec	99/99 100%	590/601 98.17%
Eigenprojection (Downsample × 16)	2.722 sec	0.0788 sec	99/99 100%	578/601 96.173%
S & K Eigenimages (Downsample × 4)	5.317 sec	0.1666 sec	96/99 96.97%	566/601 94.176%
Cartesian MSE (Downsample × 4)	11.241 sec	0.9567 sec	99/99 100%	595/601 99.002%
Cartesian MSE (Downsample × 8)	4.115 sec	0.3229 sec	99/99 100%	594/601 98.835%
Cartesian MSE (Downsample × 16)	3.251 sec	0.1451 sec	97/99 97.98%	558/601 92.845%

Table 1. Juxtaposition of identification algorithms in speed and accuracy. As expected, accuracy decreases as speed (downsampling rate) increases. All algorithms employ identical pre-processing as outlined in Section II.

Euclidean distance computation in both Cartesian coordinates and eigenspace involve comparable numbers of operations, so the mean operation times per painting are unsurprisingly similar. Both methods also load matrices of comparable sizes, although initial loading times consume a larger portion of the eigenspace time than they do for the Cartesian total time, leading to the dominance of Cartesian identification for low decimation rates. However, once the eigenprojection matrix decreases to 66 × 4096 in sixteen-fold downsampling, the speed of matrix multiplication in Matlab quickly outpaces the setup time necessary to compare in Cartesian space.

All methods maintain nearly spotless accuracy for the training set, although different algorithms miss different paintings outside the original 3 × 33 training images. A certain set of pathologically blurry images consistently fails for every identification algorithm, but the eigenprojection method remains the most robust during extreme downsampling, likely because it relies not on precise pixel placement but rather on overall training set trends, since the algorithm erects an eigenbasis from an image stack rather than individual paintings. Stress testing strains each algorithm in different ways – Cartesian MSE is more vulnerable to uneven lighting and shadow, whereas eigenprojection falls prey to wild camera angles – but no algorithm comes close to breaking at eightfold downsampling.

VIII. Conclusion

Eigenprojection seems an even more viable solution to implement in camera-phone software since nearly all necessary computation occurs *offline*, prior to use. With a reliable eigenbasis loaded into cache memory, identification and hence information can flow on-screen instantaneously upon image capture. Furthermore, when the exhibition expands, the user need only load an updated eigenprojection matrix and eigencoordinate set into the device to maintain accurate identification. However, as the pathological images reveal, severe motion blur and wide camera angles can thwart pre-processing and ruin recognition, so this implementation applies only for face-on static photography; more dynamic applications would require more robust pre-processing. Nevertheless, the reliability of eigenprojection under even severe (sixteen-fold) decimation suggests that even smaller imaging devices can gather enough information to distinguish between a finite set of known objects. After all, even a picture without a thousand pixels is worth a thousand words.

References

[1] A.K. Jain. *Fundamentals of Digital Image Processing.* Upper Saddle River, NJ: Prentice Hall, 1989, pp. 132-180.

[2] R.C. Gonzalez, and R.E. Woods. *Digital Image Processing, Second Edition.* Upper Saddle River, NJ: Prentice Hall, pp. 675-683.

[3] R.C. Gonzalez, R.E. Woods, and S.L. Eddins. *Digital Image Processing using Matlab.* Upper Saddle River, NJ: Pearson Prentice Hall, 2004, pp. 474-483.

[4] S.E. Umbaugh. *Computer Imaging: Digital Image Analysis and Processing.* Boca Raton, FA: CRC Press, 2005.

[5] M. Petrou and P. Bosdogianni. *Image Processing: The Fundamentals.* Chichester, England: John Wiley & Sons Ltd., 2004.

[6] J.S. Lim. *Two-Dimensional Signal and Image Processing.* Upper Saddle River, NJ: Prentice Hall, 1990.

[7] R.N. Bracewell. *Fourier Analysis and Imaging.* New York: Kluwer Academic/Plenum Publishers, 2003.

[8] L. Sirovich and M. Kirby. "Low-Dimensional Procedure for the Characterization of Human Faces." Journal of the Optical Society of America A, 4(3), pp. 519-524, 1987.

[9] T. Yeh, K. Grauman, K. Tollmar, and T. Darrell. "A Picture is Worth a Thousand Keywords: Image-Based Object Search on a Mobile Platform." In *CHI* 2005, *Conference on Human Factors in Computing Systems*, Portland, OR, April 2005.

[10] G. Strang. *Introduction to Linear Algebra.* Wellesley, MA: Wellesley-Cambridge Press, 2003.

[11] B. Noble and J.W. Daniel. *Applied Linear Algebra*, *Third Edition*. Upper Saddle River, NJ: Prentice Hall, 1988.

[12] S. Boyd. *Lecture Notes for EE 263: Introduction to Linear Dynamical Systems.* Stanford, CA: Stanford University Press, Autumn 2006.

[13] B. Girod. *Lecture Notes for EE 368: Digital Image Processing*, Spring 2007.

[14] G. Takacs. Conversations, 22 May 2007, 29 May 2007.

[15] S.L. Eddins. Conversations, 25 May 2007, 30 May 2007.

Acknowledgments

We thank Professor Bernd Girod and teaching assistants Gabriel Takacs and Aditya Mavlankar for the excitement and guidance they provided throughout the quarter, not only for the project but also regarding digital image processing. This document would not be possible without their assistance.