Seeing With OpenCV

Face Recognition With Eigenface

by Robin Hewitt

PART 4

Last month's article introduced Camshift – OpenCV's built-in face tracker. This month and next, this series concludes by showing you how to use OpenCV's implementation of eigenface for face recognition.

Face recognition is the process of putting a name to a face. Once you've detected a face, face recognition means figuring out whose face it is. You won't see security level recognition from eigenface. It works well enough, however, to make a fun enhancement to a hobbyist robotics project.

This month's article gives a detailed explanation of how eigenface works and the theory behind it. Next month's article will conclude this topic by taking you through the programming steps to implement eigenface.

What is Eigenface?

Eigenface is a simple face recognition algorithm that's easy to implement. It's the first face-recognition method that computer vision students learn, and it's a standard, workhorse method in the computer vision field. Turk and Pentland published the paper that describes their Eigenface method in 1991 (Reference 3, below). Citeseer lists 223 citations for this paper – an average of 16 citations per year since publication!

The steps used in eigenface are also used in many advanced methods. In fact, if you're interested in learning computer vision fundamentals, I recommend you learn about and implement eigenface, even if you *don't* plan to incorporate face recognition into a project! One reason eigenface is so important is that the basic principles behind it – PCA and distance-based matching – appear over and over in numerous computer vision and machine learning applications.

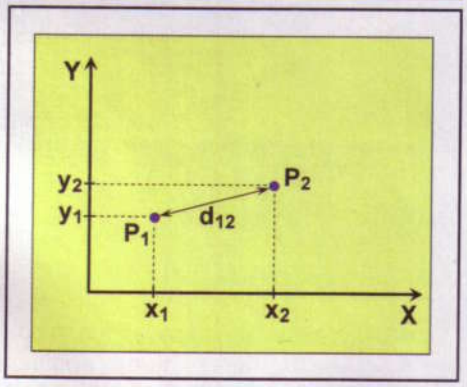
Here's how recognition works: Given example face images for each of several people, plus an unknown face image to recognize,

1) Compute a "distance" between the new image and each of the example faces.

2) Select the example image that's closest to the new one as the most likely known person.

3) If the distance to that face image is above a threshold, "recognize" the image as that person, otherwise, classify the face as an "unknown" person.

FIGURE 1. Euclidean distance, d_{12}, for two points in two dimensions.



How "Far Apart" Are These Images?

Distance – in the original eigenface paper – is measured as the point-to-point distance. This is also called Euclidean distance. In two dimensions (2D), the Euclidean distance between points P_1 and P_2 is

$d_{12} = \text{sqrt}(\Delta x^2 + \Delta y^2)$,
where $\Delta x = x_2 - x_1$, and $\Delta y = y_2 - y_1$.

In 3D, it's $\text{sqrt}(\Delta x^2 + \Delta y^2 + \Delta z^2)$. Figure 1 shows Euclidean distance in 2D.

In a 2D plot such as Figure 1, the dimensions are the X and Y axes. To get 3D, throw in a Z axis. But what are the dimensions for a face image?

The simple answer is that eigenface considers each pixel location to be a separate dimension. But there's a catch ...

The catch is that we're first going to do something called *dimensionality reduction*. Before explaining what that is, let's look at why we need it.

Even a small face image has a lot of pixels. A common image size for face recognition is 50 x 50. An image this size has 2,500 pixels. To compute the Euclidean distance between two of these images, using pixels as dimensions, you'd sum the square of the brightness difference at each of the 2,500 pixel locations, then take the square root of that sum.

There are several problems with this approach. Let's look at one of them – signal-to-noise ratio.

Noise Times 2,500 is a *Lot* of Noise

By computing distance between face images, we've replaced 2,500 differences between pixel values with a single value. The question we want to consider is, "What effect does noise

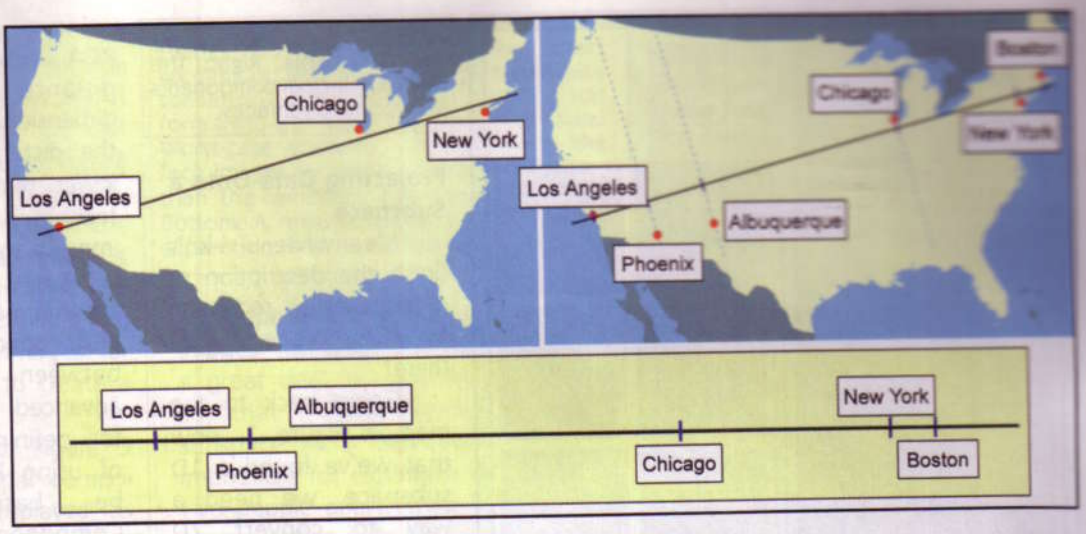


FIGURE 2. Right: Fitting a line to three points is a special case of PCA. Left: To project points from the 2D map onto the 1D line, locate the point on the line that's closest to the 2D point. Bottom: The 1D subspace, and the distances between points in this subspace.

have on this value?"

Let's define noise as anything – other than an identity difference – that affects pixel brightness. No two images are exactly identical, and small, incidental influences cause changes in pixel brightness. If each one of these 2,500 pixels contributes even a small amount of noise, the sheer number of noisy pixels means the total noise level will be very high.

Amidst all these noise contributions, whatever information is useful for identifying individual faces is presumably contributing some sort of consistent signal. But with 2,500 pixels each adding some amount of noise to the answer, that small signal is hard to find and harder to measure.

Very often, the information of interest has a much lower dimensionality than the number of measurements. In the case of an image, each pixel's value is a measurement. Most likely, we can (somehow) represent the information that would allow us to distinguish between faces from different individuals with a much smaller number of parameters than 2,500. Maybe that number is 100; maybe it's 12. We don't claim to know in advance what it is, only that it's probably much smaller than the number of pixels.

If this assumption is correct, summing all the squared pixel differences would create a noise contribution that's extremely high compared to the useful information. One goal of dimensionality reduction is to tone down the noise level, so the important information can come through.

Dimensionality Reduction by PCA

There are many methods for dimensionality reduction. The one that eigenface uses is called *Principal Components Analysis* – PCA for short.

Line Fitting and PCA

To get an intuition for what PCA does, let's look at a special case of PCA called a "least squares line fit." The lefthand side of Figure 2 shows an example of fitting a line to three points: the 2D map locations for Los Angeles, Chicago, and New York. (To keep the explanation simple, I've ignored 3D factors such as elevation and the curvature of the Earth.)

These three map points are almost – but not quite – on a single line already. If you were planning a trip, that relationship would be useful information. In that sense, a single line expresses something essential about their relationship. A line has only one dimension, so if we replace the points' 2D locations with locations along a single line, we'll have reduced their dimensionality.

Because they're almost lined up already, a line can be fitted to them with little error. The error in the line fit is measured by adding together the square of the distance from each point to the line. The best-fit line is the one that has the smallest error.

Defining a Subspace

Although the line found above is a 1D object, it's located inside a larger, 2D space, and has an orientation (its slope). The slope of the line expresses something important about the three points. It indicates the direction in which they're spread out the most.

If we position a rectangular (x,y) coordinate system so that its origin is somewhere on this line, we can write the line equation as simply

$y = \mathbf{m}x,$

where **m** is the line's slope: $\Delta y/\Delta x$.

When it's described this way, the line is a *subspace* of the 2D space defined by the (x,y) coordinate system. This description emphasizes the aspect of the data we're interested in, namely the direction that keeps these points most separated from one another.

The PCA Subspace

This direction of maximum separation is called the first principal component of a dataset. The direction with the next largest separation is the one perpendicular to this. That's the second principal component. In a 2D dataset, we can have at most two principal components.

Since the dimensionality for images is much higher, we can have more principal components in a dataset made up of images.

However, the number of principal components we can find is also limited by the number of data points. To see why that is, think of a dataset that consists of just one point. What's the direction of maximum separation for this dataset? There isn't one, because there's nothing to separate. Now consider a dataset with just two points. The line connecting these two points is the first principal component. But there's no second principal component, because there's nothing

FIGURE 3. Left: Face images for 10 people. Right: The first six principal components viewed as eigenfaces.

more to separate: both points are fully on the line.

We can extend this idea indefinitely. Three points define a plane, which is a 2D object, so a dataset with three data points can never have more than two principal components, even if it's in a 3D, or higher, coordinate system. And so on.

In eigenface, each 50 x 50 face image is treated as one data point (in a 2,500 dimensional "space"). So the number of principal components we can find will never be more than the number of face images minus one.

Although it's important to have a conceptual understanding of what principal components are, you won't need to know the details of how to find them to implement eigenface. That part has been done for you already in OpenCV. I'll take you through the API for that in next month's article.

Projecting Data Onto a Subspace

Meanwhile, let's finish the description of dimensionality reduction by PCA. We're almost there!

Going back to the map in Figure 2, now that we've found a 1D subspace, we need a way to convert 2D points to 1D points. The process for doing that is called projection. When you project a point onto a subspace, you assign it the subspace location that's closest to its location in the higher dimensional space. That sounds messy and complicated, but it's neither. To project a 2D map point onto the line in Figure 2, you'd find the point on the line that's closest to that 2D point. That's its projection.

There's a function in OpenCV for projecting points onto a subspace, so again, you only need a conceptual understanding. You can leave the algorithmic details to the library.

The blue tic marks in Figure 2 show the subspace location of the three cities that defined the line. Other 2D points can also be projected onto this line. The righthand side of Figure 2 shows the projected locations for Phoenix, Albuquerque, and Boston.

Computing Distances Between Faces

In eigenface, the distance between two face images is the Euclidean distance between their projected points in a PCA subspace, rather than the distance in the original 2,500 dimensional image space. Computing the distance between faces in the lower dimensional subspace is the technique that eigenface uses to improve the signal-to-noise ratio.

Many advanced face recognition techniques are extensions of this basic concept. The main difference between eigenface and these advanced techniques is the process for defining the subspace. Instead of using PCA, the subspace might be based on Independent Component Analysis (ICA) or on Linear Discriminant Analysis (LDA), and so on.

As mentioned above, this basic idea — dimensionality reduction followed by distance calculation in a subspace — is widely used in computer vision work. It's also used in other branches of AI. In fact, it's one of the primary tools for managing complexity and for finding the patterns hidden within massive amounts of real world data.

Picturing the Principal Components

In our definition of a line as a 1D subspace, we used both *x* and *y* coordinates to define **m**, its 2D slope. When **m** is a principal component for a set of points, it has another name. It's an eigenvector. As you no doubt guessed, this is the basis for the name "eigenface." Eigenvectors are a linear algebra concept. That concept is important to us here only as an alternative name for principal components.

For face recognition on 50 x 50 images, each eigenvector represents the slope of a line in a 2,500 dimensional space. As in the 2D case, we need all 2,500 dimensions to define the slope of each line. While it's impossible to visualize a line in that many dimensions, we can view the eigenvectors in a different way. We can convert their 2,500 dimensional "slope" to an image simply by placing each value in its corresponding pixel location. When we do that, we get facelike images called — guess what — eigenfaces!

FIGURE 4. Face images from two individuals. Each individual's face is displayed under four different lighting conditions. The variability due to lighting here is greater than the variability between individuals.

Eigenface tends to confuse individuals when lighting effects are strong.

Eigenfaces are interesting to look at, and give us some intuition about the principal components for our dataset. The lefthand side of Figure 3 shows face images for 10 people. These face images are from the Yale Face Database B (References 4 and 5). It contains images of faces under a range of lighting conditions. I used seven images for each of these 10 people to create a PCA subspace.

The righthand side of Figure 3 shows the first six principal components of this dataset, displayed as eigenfaces. The eigenfaces often have a ghostly look, because they combine elements from several faces. The brightest and the darkest pixels in each eigenface mark the face regions that contributed most to that principal component.

Limitations of Eigenface

The principal components that PCA finds are the directions of greatest variation in the data. One of the assumptions in eigenface is that variability in the underlying images corresponds to differences between individual faces. This assumption is, unfortunately, not always valid. Figure 4 shows faces from two individuals. Each individual's face is displayed under four different lighting conditions.

These images are also from the Yale Face Database B. In fact, they're face images for two of the 10 people shown in Figure 3. Can you tell which ones are which? Eigenface can't. When lighting is highly variable, eigenface often does no better than random guessing would.

Other factors that may "stretch" image variability in directions that tend to blur identity in PCA space include changes in expression, camera angle, and head pose.

Figure 5 shows how data distributions affect eigenface's performance. The best case for eigenface is at the top of Figure 5. Here, images from two individuals are clumped into tight clusters that are well separated from one another. That's what you hope will happen. The middle panel in Figure 5 shows what you hope *won't* happen. In this panel, images for each individual contain a great deal of variability. So much so, that they've skewed the PCA subspace in a way that makes it impossible for eigenface to tell these two people apart. Their face images are projecting onto the same places in the PCA subspace.

In practice, you'll probably find that the data distributions for face images fall somewhere in between these extremes. The bottom panel in Figure 5 shows a realistic distribution for eigenface.

Since the eigenvectors are determined only by data variability, you're limited in what you can do to control how eigenface behaves. However, you can take steps to limit, or to otherwise manage, environmental conditions that might confuse it. For example, placing the camera at face level will reduce variability in camera angle.

Lighting conditions – such as side lighting from windows – are harder for a mobile robot to control. But you might consider adding intelligence on top of face recognition to compensate for that. For example, if your robot knows roughly where it's located, and which direction it's facing, it can compare the current face image only to ones it's seen previously in a similar situation.

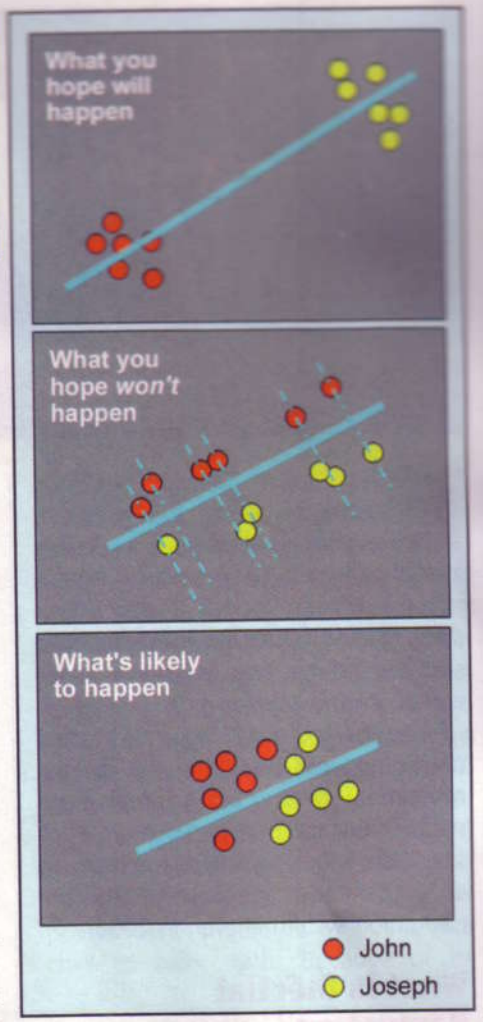
Even highly-tuned commercial face recognition systems are subject to cases of mistaken identity. In fact, part of the challenge of incorporating face recognition into any robotics application is finding ways to accommodate these.

Coming Up

Next month's article concludes this series by taking you step-by-step through a program that implements eigenface with OpenCV.

Be seeing you! SV

FIGURE 5. How data distributions affect recognition with eigenface. Top: Best-case scenario – data points for each person form tight, well separated clusters. Middle: Worst-case scenario – variability in the face images for each individual is greater than the variability between individuals. Bottom: A realistic scenario – reasonable separation, with some overlap.



References and Resources

- *OpenCV on Sourceforge* **http://sourceforge.net/projects/opencvlibrary**
- *Official OpenCV usergroup* **http://tech.groups.yahoo.com/group/OpenCV**
- Turk, M., Pentland, A., *Face recognition using eigenfaces*, Proc. IEEE Conf. on Computer Vision and Pattern Recognition, 1991.
- *Yale Face Database B* **http://cvc.yale.edu/projects/yalefacesB/yalefacesB.html**
- A.S. Georghiades, P.N. Belhumeur, D.J. Kriegman, D.J.,*From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose*, IEEE Trans. Pattern Anal. Mach. Intelligence, 2001.