Video: Early Face Recognition Methods

[00:01] [slide 1] This second video introduces two early approaches to the computer recognition of faces that had a strong influence in the field. The first is where it all began, with Takeo Kanade, who built the first end-to-end automated face recognition system around the time I was graduating from high school, and the second is a method proposed by Turk and Pentland that goes by the name of “Eigenfaces.”

[00:27] Takeo Kanade went on to be a giant in the field of computer vision, and a photo of him appears in the image in the bottom left corner, highlighted with the red box. The image shows the results of a later system he built for detecting faces in an image. For his PhD thesis at Kyoto University in 1973, he built a system that started with a digital image. He built his own digitizer to scan in images that were 140x200 pixels and had 32 shades of gray, which was a large image in those days. He processed the images to find edges, as shown in the upper right corner. The system then applied lots of special purpose algorithms to locate particular features in the image like certain points on the eyes, nose, mouth, top of the head, boundaries of the face, and so on. About 40 geometric quantities were then computed from these features. These quantities are shown on the cartoon face in the bottom right corner, and they included ratios of distances between feature points, and angles like the orientation of the jaw line. A vector of all these geometric measurements served as a signature to identify an individual person, which could be used to recognize that person in a new image. Why did he use relative distances rather than absolute distances, for these geometric features? There could be variation in the size of the face in the image, and relative distances enabled the system to handle changes in scale.

[02:16] This was quite a technical feat at the time - he constructed a database of 800 images of 20 different people, and his system recognized people correctly about 75% of the time. This general approach has been used in many face recognition systems in the years since. [slide 2] An early version of the system became quite famous and was featured in a public attraction at the 1970 World Expo held in Osaka, Japan. In advance, Kanade scanned photos of a lot of famous people like Marilyn Monroe, JFK, and Winston Churchill, and so on, and he analyzed these photos to get a representation of their geometric features. At a booth at the Expo, he took a photo of each visitor, analyzed it to compute the geometric features for this new person, and compared it to those of the famous people to find who was the best fit, for example was the person a “Winston Churchill type,” and the visitors were given a souvenir photograph of their “doppelganger. ”

[03:30] [slide 3] Continuing our tour of early approaches to face recognition in computer vision, we’ll explore a classic approach championed by Turk and Pentland in the early 90s - it’s known as the Eigenface approach, and it’s based on principal components analysis that some of you may be familiar with. At some level, we think of an image as a two-dimensional thing, but we can also think of an image as a very high-dimensional thing, where each image pixel is a dimension that can take on a range of values corresponding to the

brightness at that pixel. Much of the information stored in individual pixels may be redundant, so for the purpose of representing the image for recognition, it would be nice if we could take this very high-dimensional data, and map it onto a lower dimensional space that removes a lot of the redundancy that exists in an image. Effectively we're creating a more compact representation of face images that still preserves the essential information content that distinguishes different individuals. PCA is one technique that can be used to construct this lower-dimensional representation.

[04:50] To provide some intuition about the method, we'll start with a very simple example. Imagine that you take lots of images and measure the brightness at two specific pixels that are close to one another, like the two red pixels in the image snippet in the middle. Then across all the images, you plot all combinations of the two brightnesses that you measured. If the pixels were nearby, chances are they're highly correlated, giving rise to a plot like the one shown in the lower right corner. For all the data points here, the brightnesses of pixels 1 and 2 from the same image, are similar to one another, which you'd expect in typical natural images. Most of the spread in this data occurs along the direction of the red line here on the graph. That red line is the direction of the first principal component, it's the direction that captures the most variation in the data. So if you want to capture what's most important in these two pixels, you could take each point that's initially defined by the two separate pixel values, and project it onto this one dimension in red, and just store one value for each pair, which is the location of the projection along this direction of the first component. There is variation in the data in the direction perpendicular to the first component, and that's captured in the second principal component that's shown in green here. But for the task that you're using the information for, it may be adequate just to preserve information about where the pair of pixels lie along the first component. So we reduced two dimensions of data to one.

[06:53] [slide 4] Applying PCA to images starts with a training set of images, like this sample set of face images from the Yale face database. A training set typically contains multiple images of each person that may have some variation, and the images are assumed to be aligned and cropped so they all have similar size and pose, and they have a simple background.

[07:20] [slide 5] To prepare the image data for PCA, we follow the three steps described here. First, for each image like the one shown enlarged here, we take each of the columns and lay them out end-to-end in one long column vector, like this. The length of this vector is the total number of pixels in the image, rows times columns, which I denote by M here. Then we place the column vectors generated from each image in the training set side-by-side in a large matrix, like this. This matrix has M rows and N columns storing each of the N images in the dataset. The last step is to compute an average face by taking an average of all the images in an element-by-element way, so each location of the matrix stores the average of all the brightnesses measured at that location, across all the images. We create a long column vector for this average face and subtract it from each column of this data matrix being constructed. Then we're set to run PCA.

[08:44] [slide 6] Here's our image data matrix again, that will be the input for PCA. The algorithm computes the eigenvectors of a covariance matrix computed from the data. I won't be going into the mathematical details here, because they're not essential to understanding how we're using PCA to represent face images, but these are terms that may be familiar to you if you've taken a linear algebra course, or an advanced statistics course. Each eigenvector is a principal component of the data, it's a vector of M elements like the columns of the data matrix, and we're going to select K principal components, where K is typically much smaller than the number of images N. The first principal component, which is stored in the first column in this resulting matrix, captures the direction of largest variation in the input data. The second component captures the direction of next largest variation in the data. This is analogous to the red and green directions of variation for the simple two-dimensional data that we started with. We can visualize each principal component as an image, by dividing up the long column vector into individual columns and stacking them side-by-side as an image. For the Yale training set, the first principal component looks like this. It vaguely resembles a face, although it looks a little creepy. Turk and Pentland referred to these as eigenfaces because of their origin and how they're computed.

[10:35] [slide 7] How are these principal components, or eigenfaces, used to represent face images in a database, and how do they enable recognition of new face images? This picture shows an array of eigenfaces created from a different training set. In the upper left corner is the average face, where again, the brightness at each location is the average of all the brightnesses at that same location across the entire stack of images. This average was generated from a database where each image had a dark background, which is why the background is in black here. In preparation for PCA, this average face was subtracted from each image in the training set. To the right of the average face is the first eigenface, and that's followed by the second and third eigenfaces along that first row. And the next rows contain later eigenfaces, in sequence, from left to right, top to bottom. The early components capture most of the variation across the dataset, and later components capture more subtle variations between face images. How do we use these components to represent an individual face image? For each image in the database, we compute a set of weights, one for each eigenface. The image, denoted here as F(x,y) is then expressed as a sum of the average face plus a weighted combination of the eigenfaces. Here, the weights are denoted by wi and the eigenfaces are E(x,y), which is also indexed by i. So a particular face is represented as the sum of the average plus a little bit of the first eigenface, plus a little bit of the second eigenface, and so on, and the weighted components are added together in an element-by-element way.

[12:58] Look for a moment at the average face in the upper left corner and the first eigenface next to it. This eigenface has a fairly uniform dark region over the extent of the face area. An individual face will have a particular weight associated with this first eigenface, and this weight could be positive or negative. The combination alone of the average face with some amount of this eigenface effectively would lighten or darken the skin tone. Other

eigenfaces lower down have face-like features such as the beard on the far right of the second row. A person who really has a beard would probably have a large weight associated with this eigenface. Adding together these eigenfaces with different weights would produce different faces, capturing the variation that exists in the particular training set.

[14:07] [slide 8] This last slide illustrates the weighted sum in a slightly different way, using the eigenfaces generated from yet another training set. Here we have a particular face image from the database shown at the top, and the set of eigenfaces are arranged along a row at the bottom, and this face image is represented as the sum of the mean or average face, plus weight w1 times the first eigenface, plus weight w2 times the second eigenface, and so on, down to weight k times the kth eigenface. As you can see here, there are these k weights, one for each of the k eigenfaces. Given the eigenfaces, computing these weights for each image in the database is a direct calculation involving a matrix multiplication. You can think of the set of weights associated with a particular image as a signature that captures that image. k is a small number compared to the M pixels in the original image - k might be about 25 for a smallish database of a few hundred face images. So each image is represented by a set of k weights rather than representing it by its original M pixels. So it's a much more compact representation.

[15:44] Now, given a new image to recognize, the first step in the recognition process is to compute a set of weights for the new face, using the eigenfaces. We can then search through the sets of weights for our known individuals in the database, to find the one that yields the closest match. There are different metrics we can use to compare two sets of weights, just like there were different measures of similarity that we could use for the matching process in stereo or motion. The simplest metric is euclidean distance, which is shown in the bottom right corner here. i is the index for each weight, and m refers to a sample individual in the database, and we measure how different are two sets of weights by taking the sum of the squared differences between the corresponding weights. This gives us a way to find identity by selecting the individual in the database whose weights minimize this difference, which is the individual whose signature weights are closest to the weights computed for the new face image.

That's the Eigenfaces approach, and it's still used today for some applications of recognition. To see results, you'll be exploring the application of this approach to the Yale face database in your next assignment.