Eigen Eyes Nov. 1, '04

References:

- trainEigenEyes.m
- detectEigenEyes.m
- ~jepson/pub/matlab/utvisToolbox/tutorials/eigenTut
- http://diwww.epfl.ch/mantra/tutorial/english/pca/html/

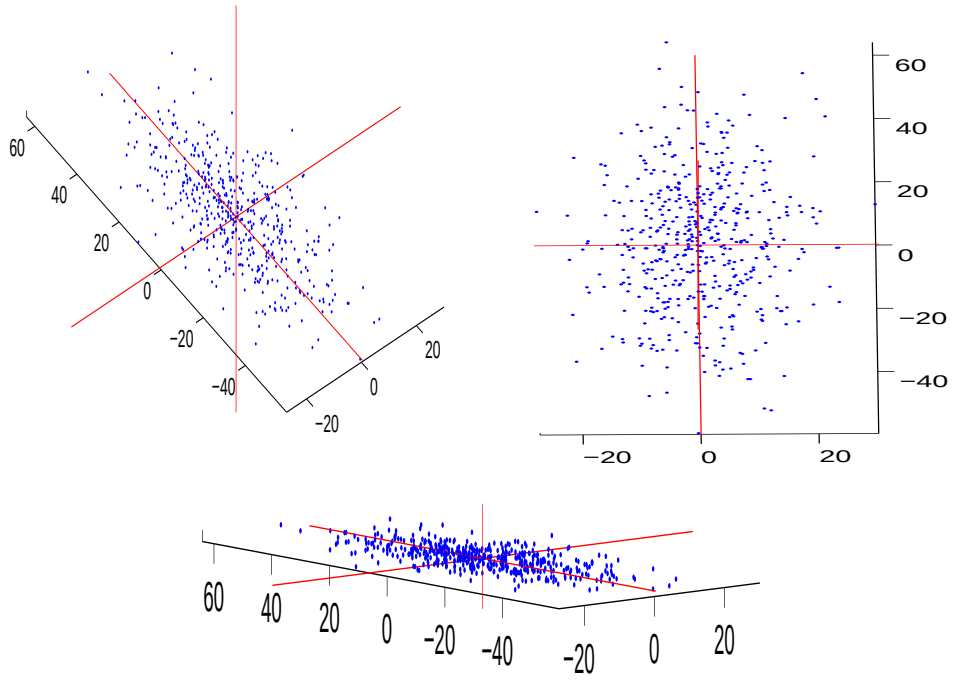
Principal Component Analysis

Principal Component Analysis is a technique that is commonly used to discover meaningful structure within a data set, it provides a representation that makes explicit similarities between data elements, and may lead to a low-dimensional representation of high-dimensional data.

This tutorial will present a small introduction of Principal Component Analysis in the context of a Gaussian blob of 3D points, and will then move on to show how PCA can be used to identify a suitable representation for a set of training images that captures interesting structure in the input data; and to perform recognition tasks (image classification).

Principal Component Analysis

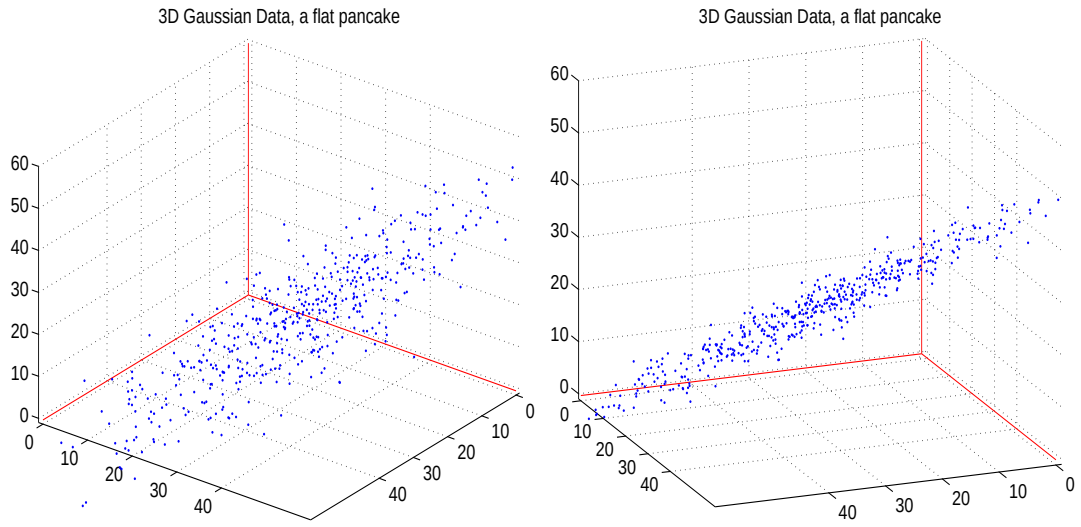
Consider the following Gaussian blob in 3D space.



Notice that the distribution is roughly elliptical in 2 of its dimensions, and quite flat along the third dimension, much like an elongated pancake.

Principal Component Analysis

Now look at the same distribution, but rotated and translated away from the origin



We will analyze the above cloud of points using PCA and see what it has to say about the structure of the data.

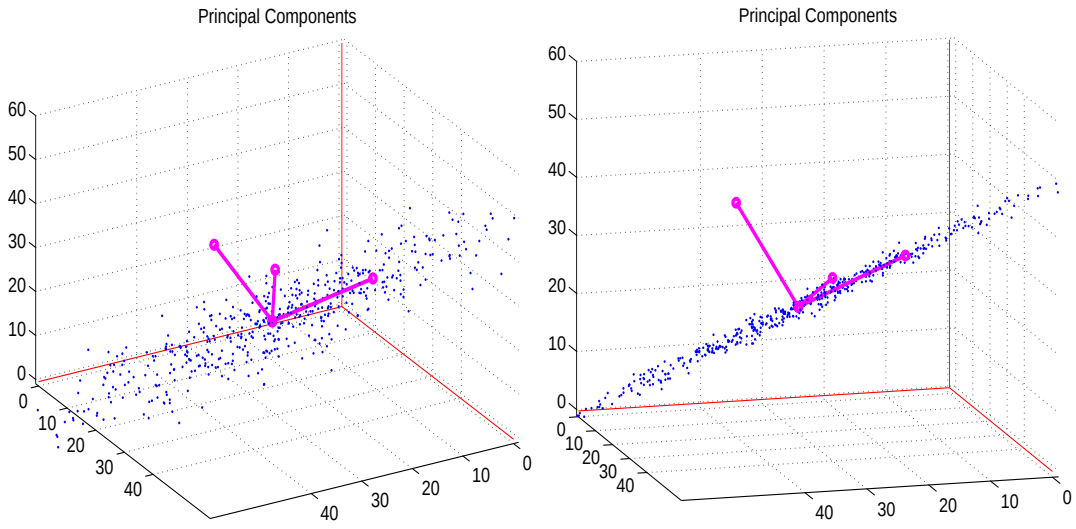
Principal Component Analysis

Given an input set of data points, all of which are row vectors of the form $\vec{x} = [a_1 a_2 \cdots a_N]$, where each a_i is an element of $\vec{x}$ and the data has dimensionality N, Principal Component Analysis consists of the following general procedure:

- Clean up input data as much as possible (remove outliers)
- Subtract the mean from the data
- Compute the covariance matrix $cov = \frac{1}{K-1} \sum_{i=1}^{K} \vec{x}^T \cdot \vec{x}$
- Compute the eigenvectors and eigenvalues of the covariance matrix (svd)
- Determine an appropriate number of basis vectors
- Project the original data onto the selected eigen-basis

Principal Component Analysis

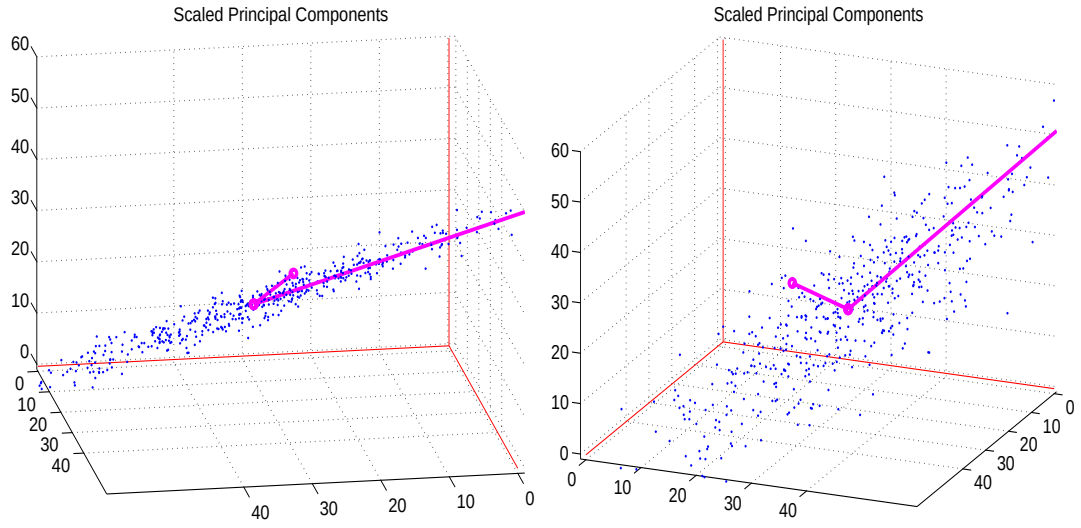
Applying the above procedure to the Gaussian data of the previous figure, we obtain the principal components shown below.



Notice that the Principal Components are oriented in the direction of the main axes of the rotated Gaussian data.

Principal Component Analysis

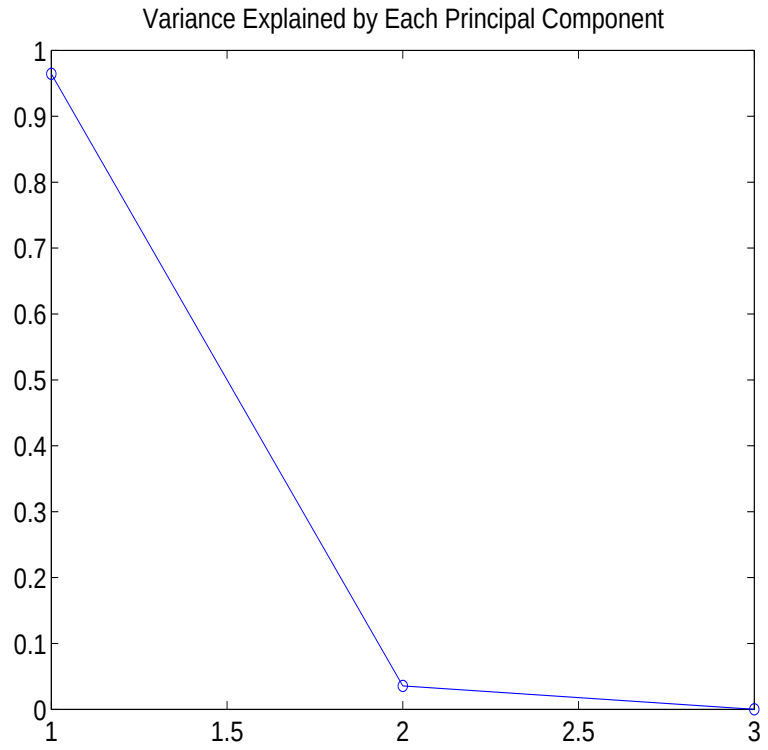
The eigenvalues of the covariance matrix indicate how much of the variance in the data set is captured by the corresponding eigenvector.



Once the eigenvectors are scaled by the corresponding eigenvalues, the structure of the input data set becomes evident.

Principal Component Analysis

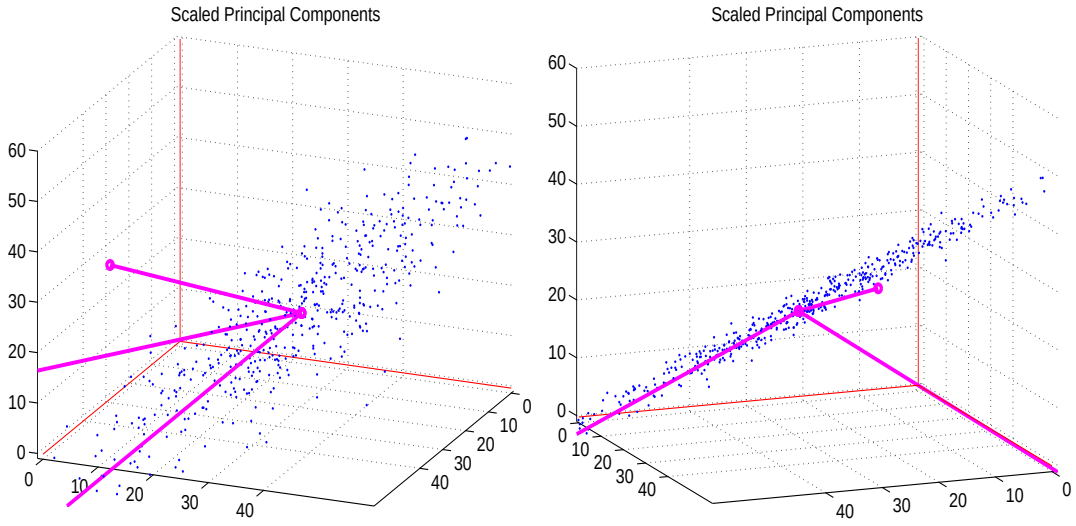
The following plots show the variance captured by each of the eigenvectors.



As expected, 2 eigenvectors capture almost all the variance in the data-set, the third one, corresponding to the flattened direction of the data set, accounts for less than 1% of the variance in the data. For more complicated data-sets, this won't be so easy.

Principal Component Analysis

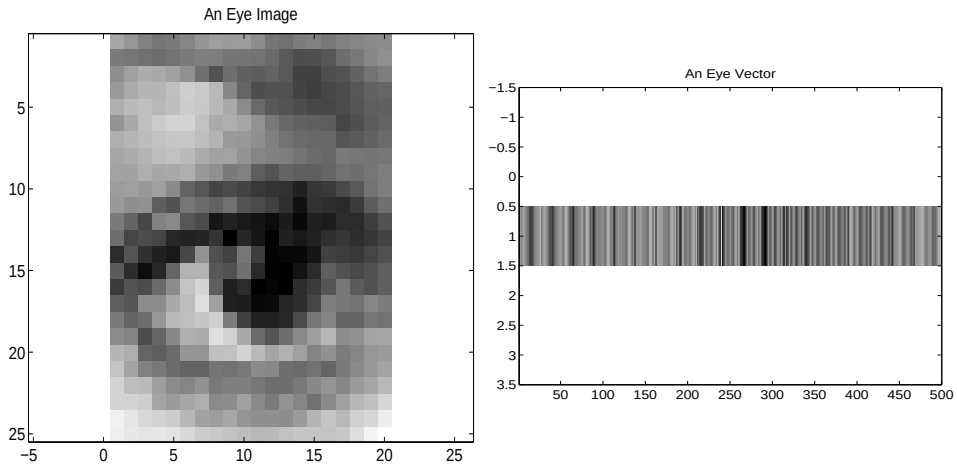
One last comment before moving on to images, the removal of the mean from the data-set is a necessary step before the calculation of the covariance matrix, otherwise the resulting eigenvectors capture something other than what we want!



The above figure shows the scaled eigenvectors that result from performing the analysis without having subtracted the mean from the original distribution. The first eigenvector now passes through the origin, and points toward the mean of the distribution. The other two vectors are forced to be orthogonal to the first one, so they are unlikely to correspond to principal directions in the data.

Principal Component Analysis

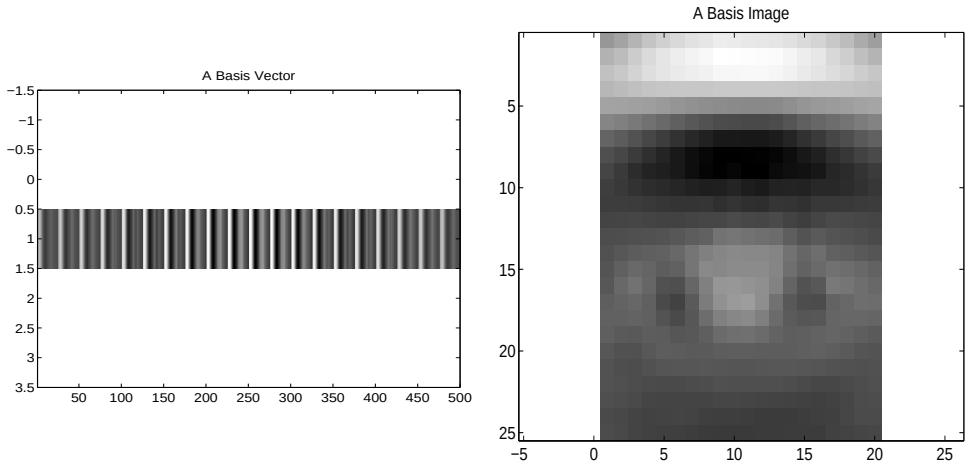
So how can we apply PCA to an image? Consider the following:



An image of $m \times n$ pixels can be interpreted as a $m * n$ dimensional vector of intensity values. Hence, an image is nothing more than a data point in a $m * n$ dimensional space. If we gather a data-set with many images of a given object (say, eyes for example), we could expect the images to have some similarity, and hence their corresponding vectors to have some underlying (non-random) distribution in space. We can apply PCA to these images to try to determine what the structure of this distribution is!

Principal Component Analysis

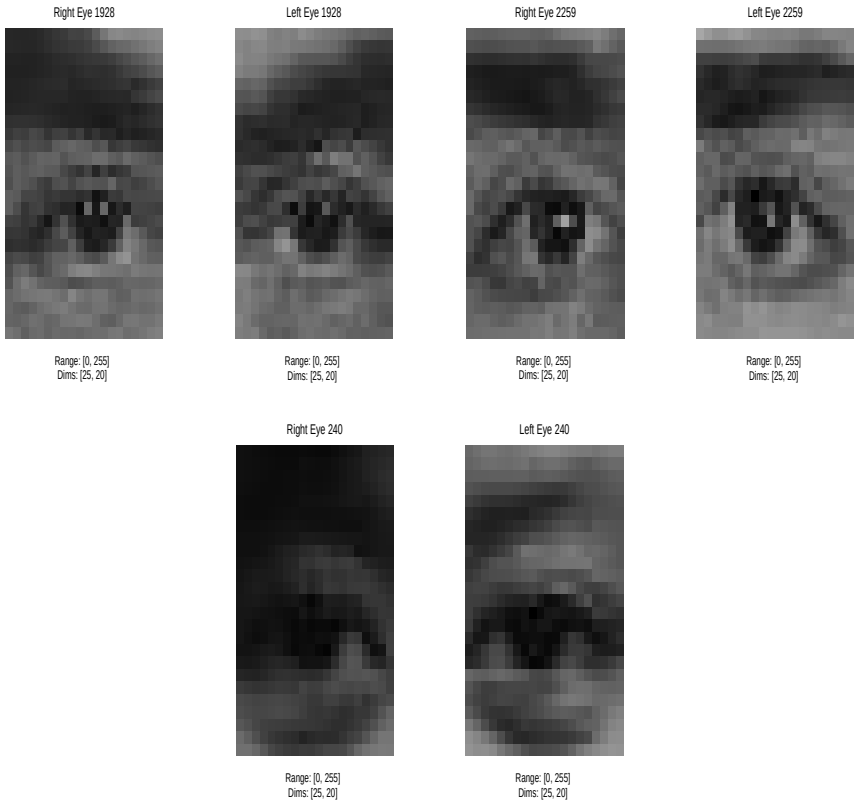
In an analogous way, each of the $m * n$ dimensional principal vectors determined by PCA can be viewed as a $m \times n$ image:



Hence, we can think of each basis vector in terms of an equivalent basis image.

Eigen Eyes

We'll now see what PCA can tell us about a set of images of eyes, the data-set contains a large number of eye pairs from many different people, under different lighting conditions, with varying contrast, and with artifacts such as eye-glasses, or images where the eyes are closed. Some sample eyes are shown below



The images have been normalized so that the eye is centered, and so that the scale of the eyes is similar from image to image.

Eigen Eyes

As we mentioned before, the first step toward PCA is to clean up the input data-set, we will use only half of the available eye images for training, and keep the other half for testing purposes.

In our particular case, we take the following steps to clean-up the data:

- Removal of the mean, and DC components of the data
- Contrast normalization
- Removal of outliers

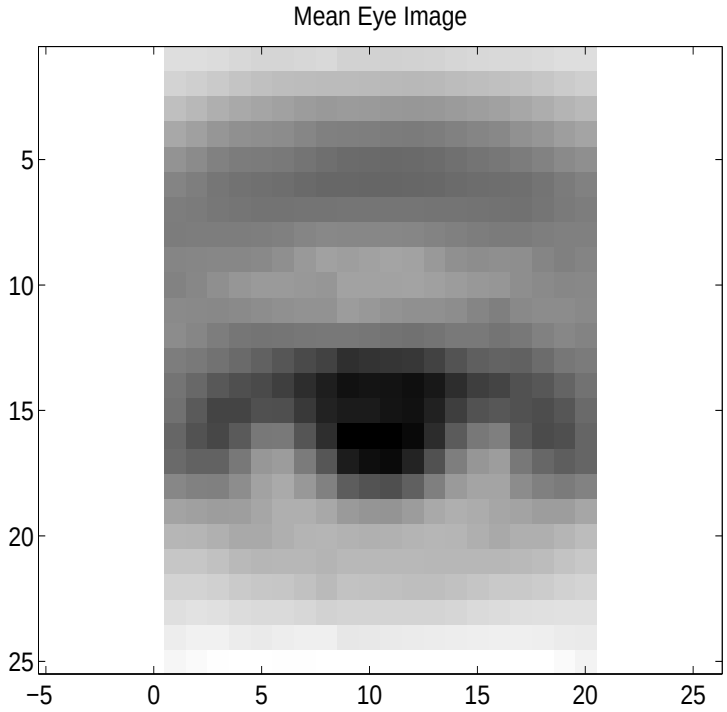
The reason for removing the mean was illustrated above, contrast normalization serves the purpose of keeping images with very high contrast from dominating the resulting PCA basis.

Outliers are removed because PCA is in effect a Least Squares estimation procedure, and we've seen before that Least Squares estimation is very sensitive to distant outliers.

Finally, the reason for keeping half of our data for testing will become apparent when we are dealing with the problem of choosing a suitable eigen-basis for our eye images.

Eigen Eyes

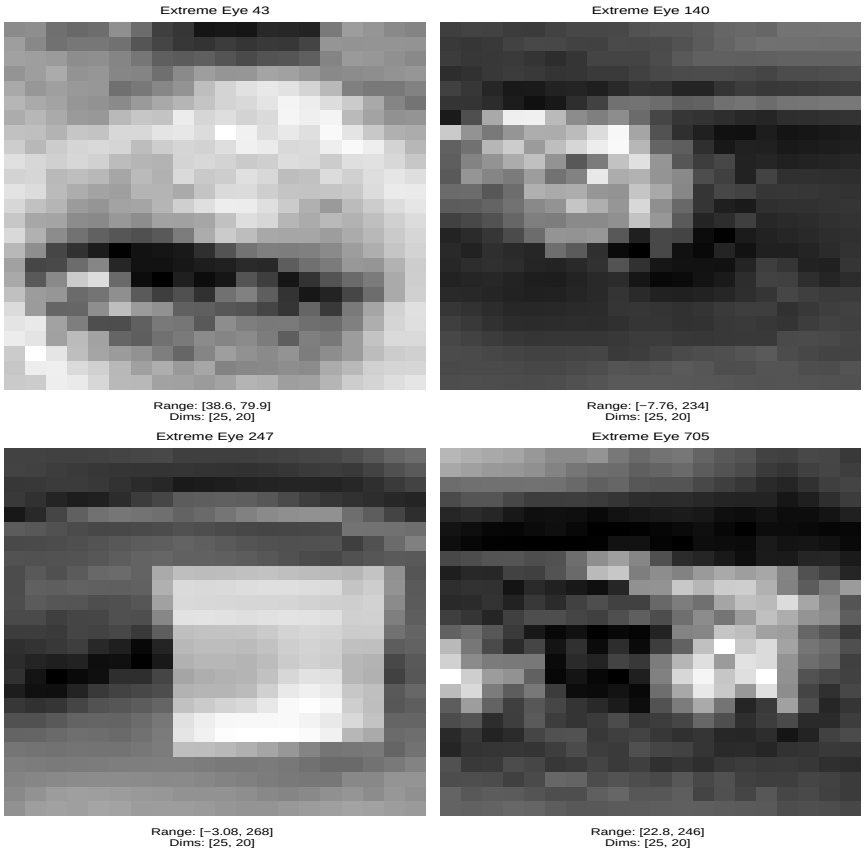
The figure below shows the mean image computed from the training data set.



This is what an average eye looks like! (according to our training data, but keep in mind that we have not removed the outliers). Each eye image in our training set can be though of as a departure from the mean eye, that is, the mean eye image accounts for a significant portion of the variance in each of the eye images in the training data. We want to use PCA to characterize and account for the remaining variance, but first we must remove outliers.

Eigen Eyes

To remove outliers, we eliminate any images in the training set that after contrast normalization and rescaling still have a large amount of variance per-pixel.

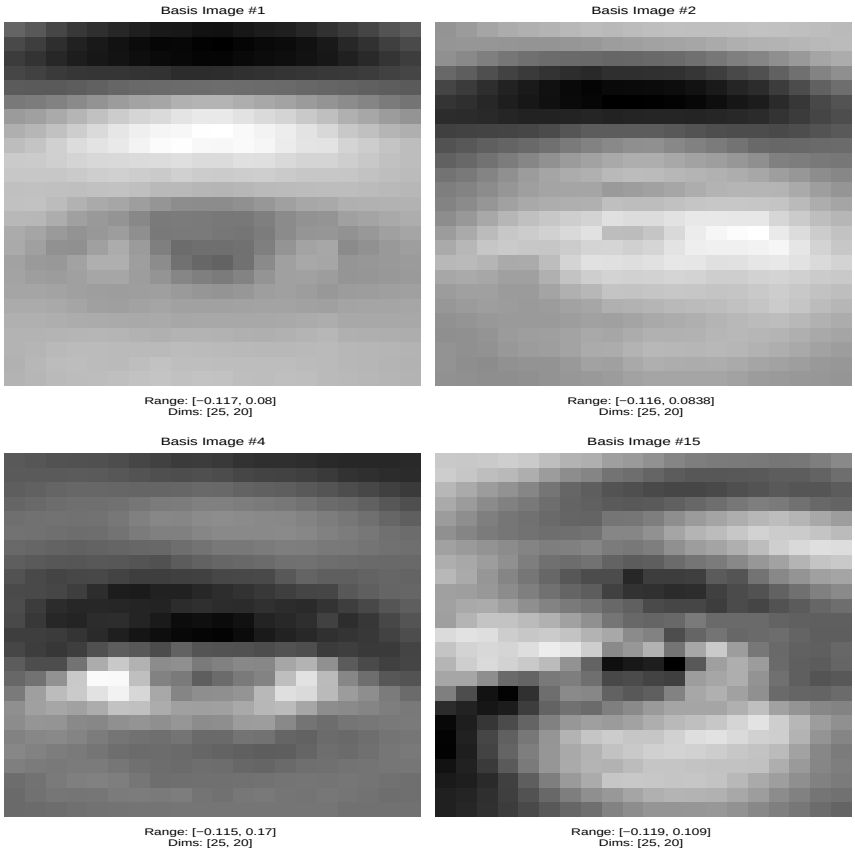


The above images show some of the outliers removed from the training data-set, they usually show closed eyes, eyes with highlights due to eye-glasses, and other cases that have a

significant deviation from the average eye. After removing outliers, and re-calculating the mean of the remaining training images, create a matrix whose columns are the training images, and perform svd on it.

Eigen Eyes

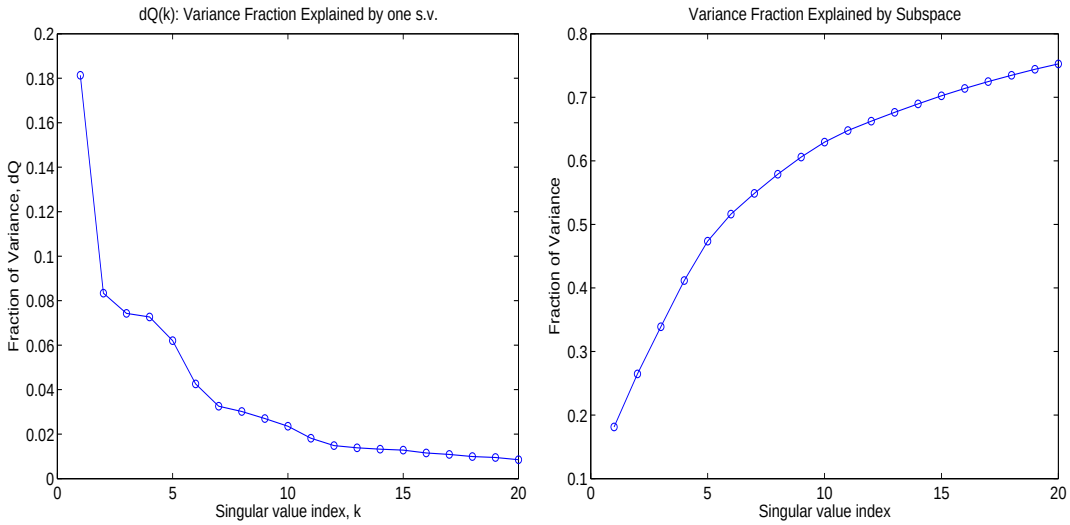
From svd, we obtain a set of basis images that capture the structure of our training data:



The first few eigenvectors capture large-scale structure in the training images, subsequent eigenvectors capture structure at progressively finer scale.

Eigen Eyes

We can look at a plot of the variance captured by each of the first few basis images.



Notice that a relatively small number of basis images captures a large portion of the variance in the training set (keep in mind that we have in total 500 eigenvectors for the eye images).

Eigen Eyes

Given the eigen-basis obtained from svd, we can represent any image $A_k(\vec{x})$ in our training set as a linear combination of basis-images:

$$\hat{A}_k(\vec{x}) = \sum_{j=1}^{M} c_{k,j} B_j(\vec{x}) \tag{1}$$

Where M is the dimension of the chosen sub-space that will be used to represent the training data, $c_{k,j}$ is the coefficient for image k and basis image j given by $c_{k,j} = \langle A_k(\vec{x}), B_j(\vec{x}) \rangle$, and $\frac{1}{K} \sum_{k=1}^{K} c_{k,j}^2 = \sigma_j^2$.

The σ_j are related to the singular values of the matrix we performed svd on by the following equation: $\sigma_j = \frac{1}{K} \kappa_j$, where κ_j are the singular values we get from svd.

We can calculate the reconstruction error for an image $E_k(\vec{x}) = A_k(\vec{x}) - \hat{A}_k(\vec{x})$, the per-pixel variance $V_n(\vec{x}) = \frac{1}{K} \sum_{k=1}^{K} E_k^2(\vec{x})$, and two useful statistics:

$$S_k^{wis} = \sum_{j=1}^{M} \frac{c_{k,j}^2}{\sigma_j^2} \tag{2}$$

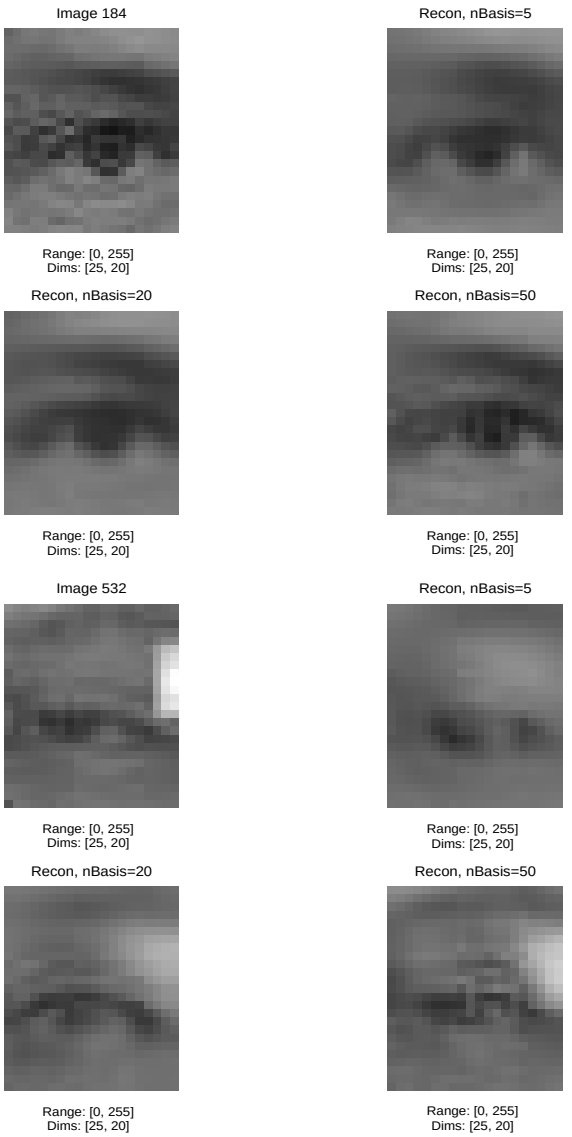
is the $within\ subspace\ statistic$ and measures the magnitude of the coefficients for image k with regard to the expected variance of the basis images, and

$$S_k^{oos} = \frac{1}{N} \sum_{\vec{x}} \frac{E_k^2(\vec{x})}{V_n(\vec{x})} \tag{3}$$

which measures the unexplained variance of the image as a fraction of the expected per-pixel variance. S_k^{oos} is known as the $out\ of\ subspace\ error$.

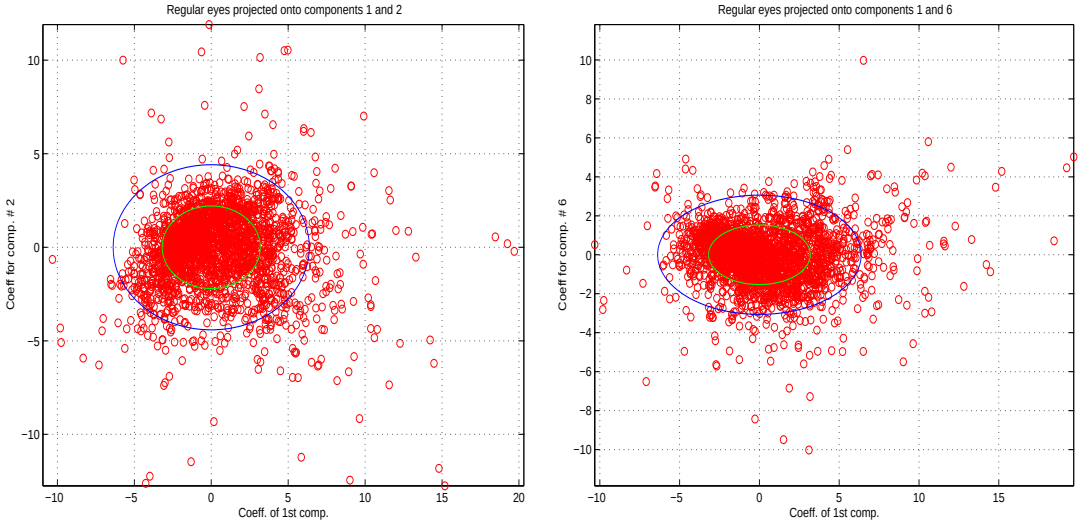
Eigen Eyes

The figure below shows sample training images, and their reconstruction using subspaces of different dimension.



Eigen Eyes

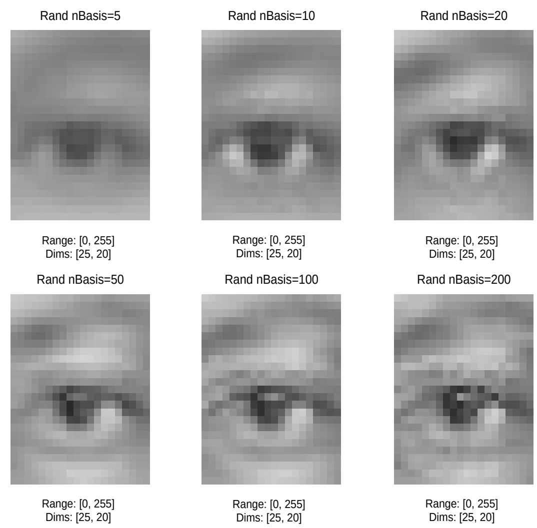
We can go a bit further than this, it turns out that the distribution of coefficients for each basis image can be approximated with a Gaussian PDF.

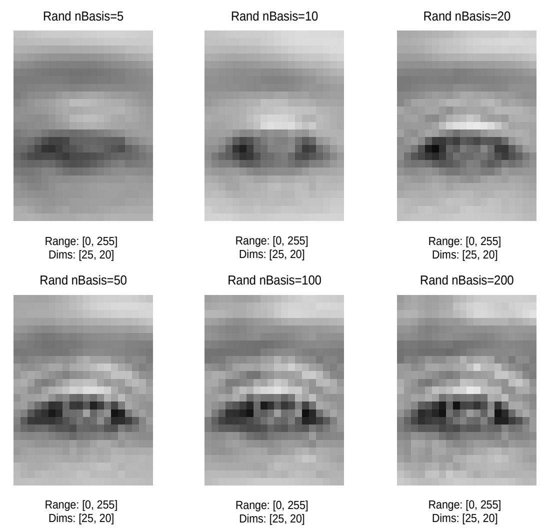


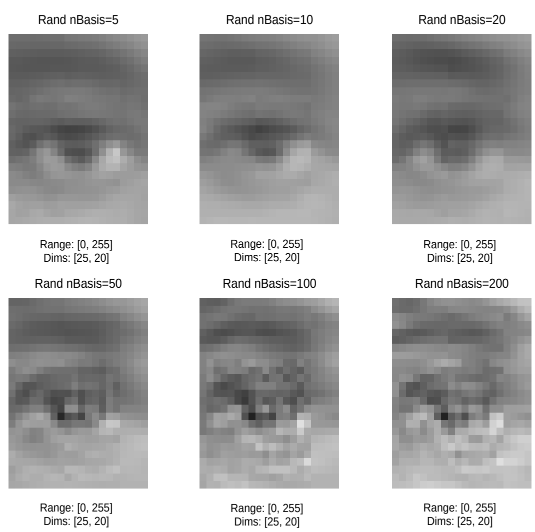
Hence, we can 'hallucinate' eyes by generating coefficients for each basis from the appropriate Gaussian PDF and performing the reconstruction process.

Eigen Eyes

The following images show some of the 'eyes' that can be generated from the basis images in this manner.







Eigen Eyes

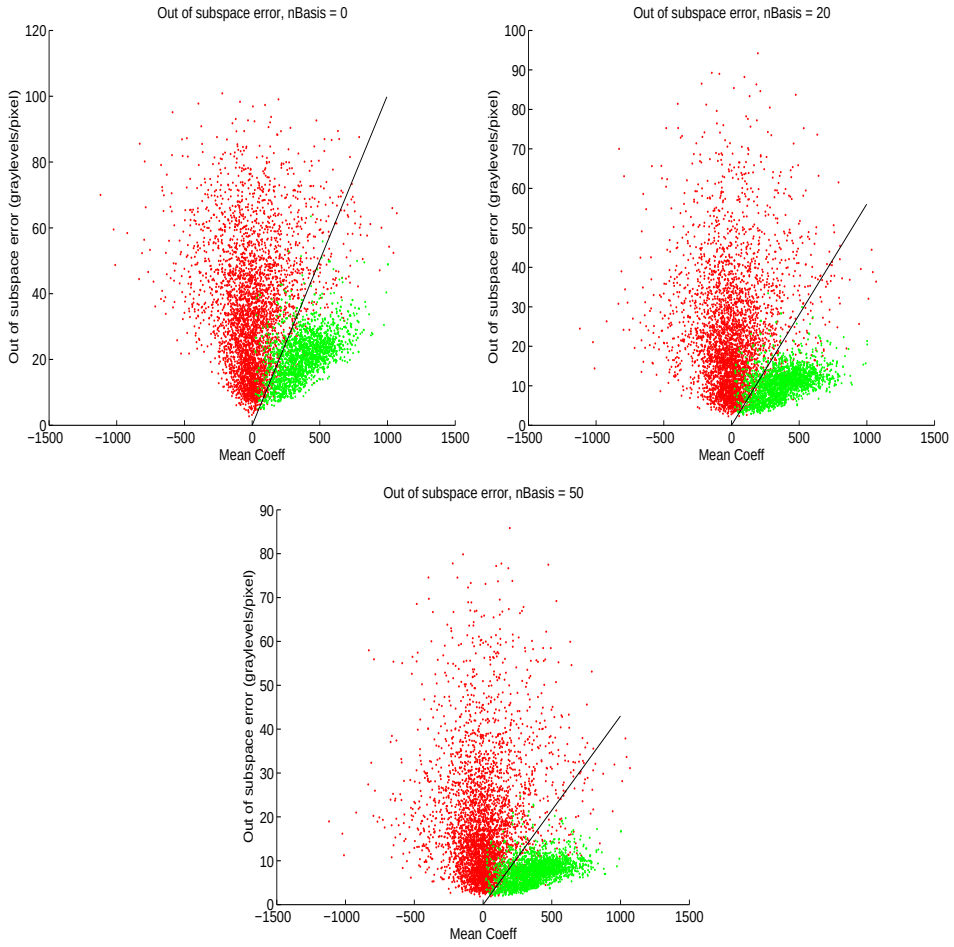
Given the above basis images, we can represent our original training data in a much more compact way, instead of storing each complete $m \times n$ image, we need only store the M basis images we have chosen for our subspace, the average eye image, and M coefficients to represent each training image with the chosen eigen basis.

However, this is not all that we can do. It is possible to use the eigen-basis to detect new instances of eyes that we haven't seen before in our training set. And given a set of images that contains eyes and non-eyes (image patches that contain some other structure), we can classify with relatively good accuracy the images that contain eyes and the images that contain non-eyes.

The detection process is based on the error metrics described above, in particular, the S_k^{oos} metric, which corresponds to the 'out of subspace' error, and quantifies the variance in an image that is not accounted for by our chosen eigen-basis.

Eigen Eyes

Recall that we saved half of our eye data-set for testing, we can use that half of the data-set, plus a data-set of non-eye images to characterize the performance of our eigen-basis for classification.

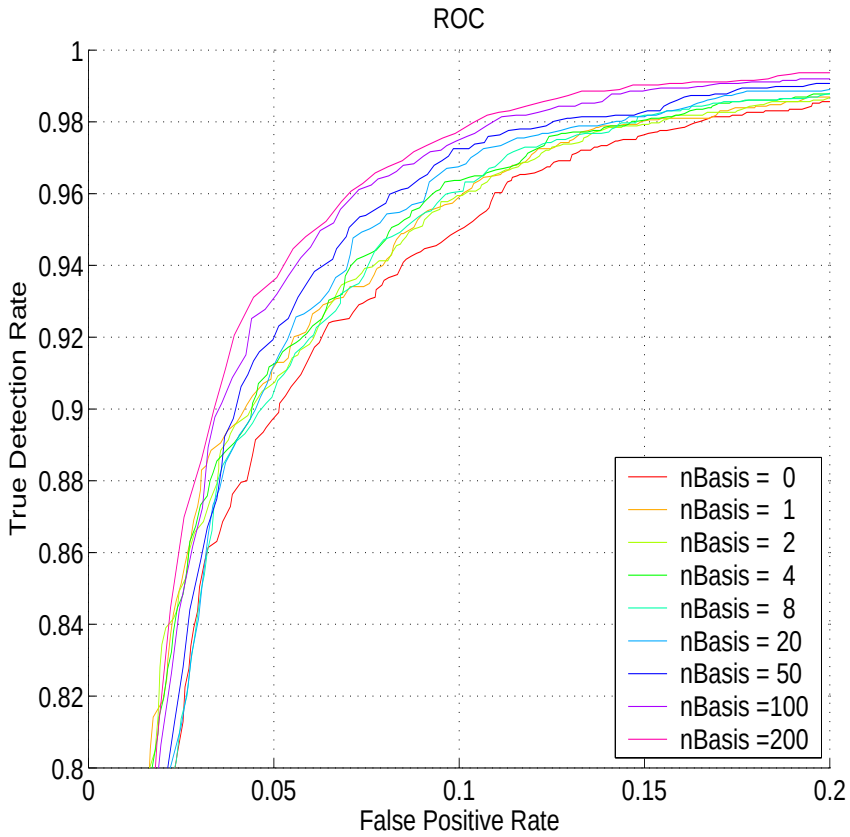


The above figures show the out of subspace error for eyes (green) and non-eyes (red) given sub-spaces of different dimension. The

classification criteria is simple, any point below and to the left of the line is classified as an eye. Notice that given the mixture of the eyes and non-eyes close to the boundary between the classes, it is not possible to obtain perfect classification.

Eigen Eyes

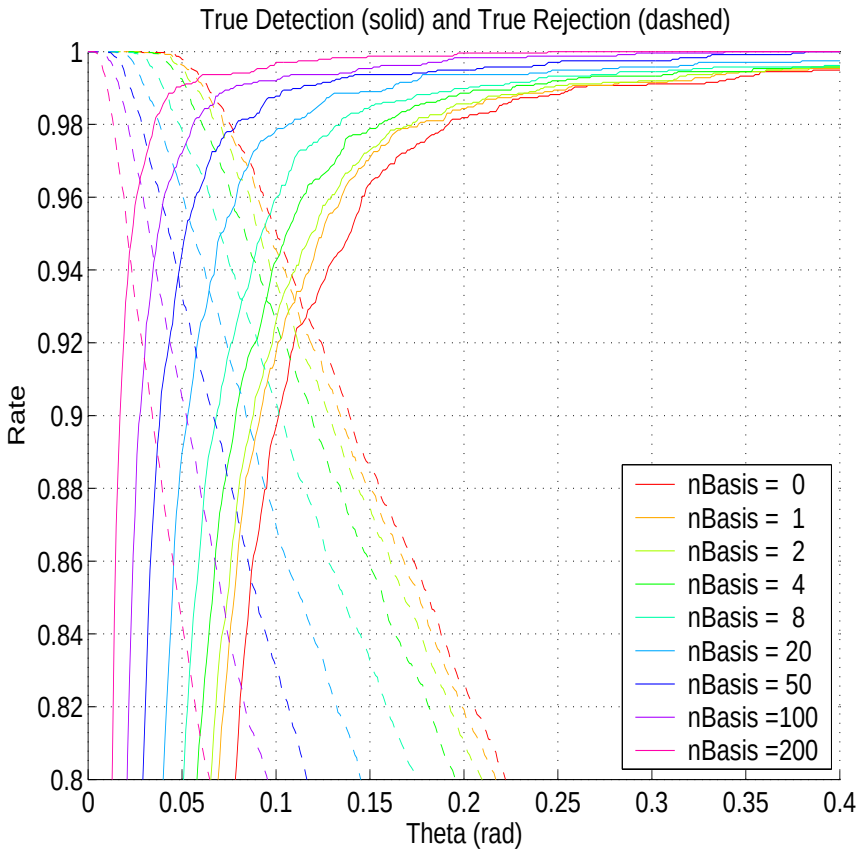
We can also characterize the performance of our eigen-basis for classification by looking at the ROC curve (Receiver Operating Characteristic), which plots the true-detection rate against the false-positive rate.



Notice that after the subspace has grown to about 50 basis images, adding more basis images does not produce a significant improvement.

Eigen Eyes

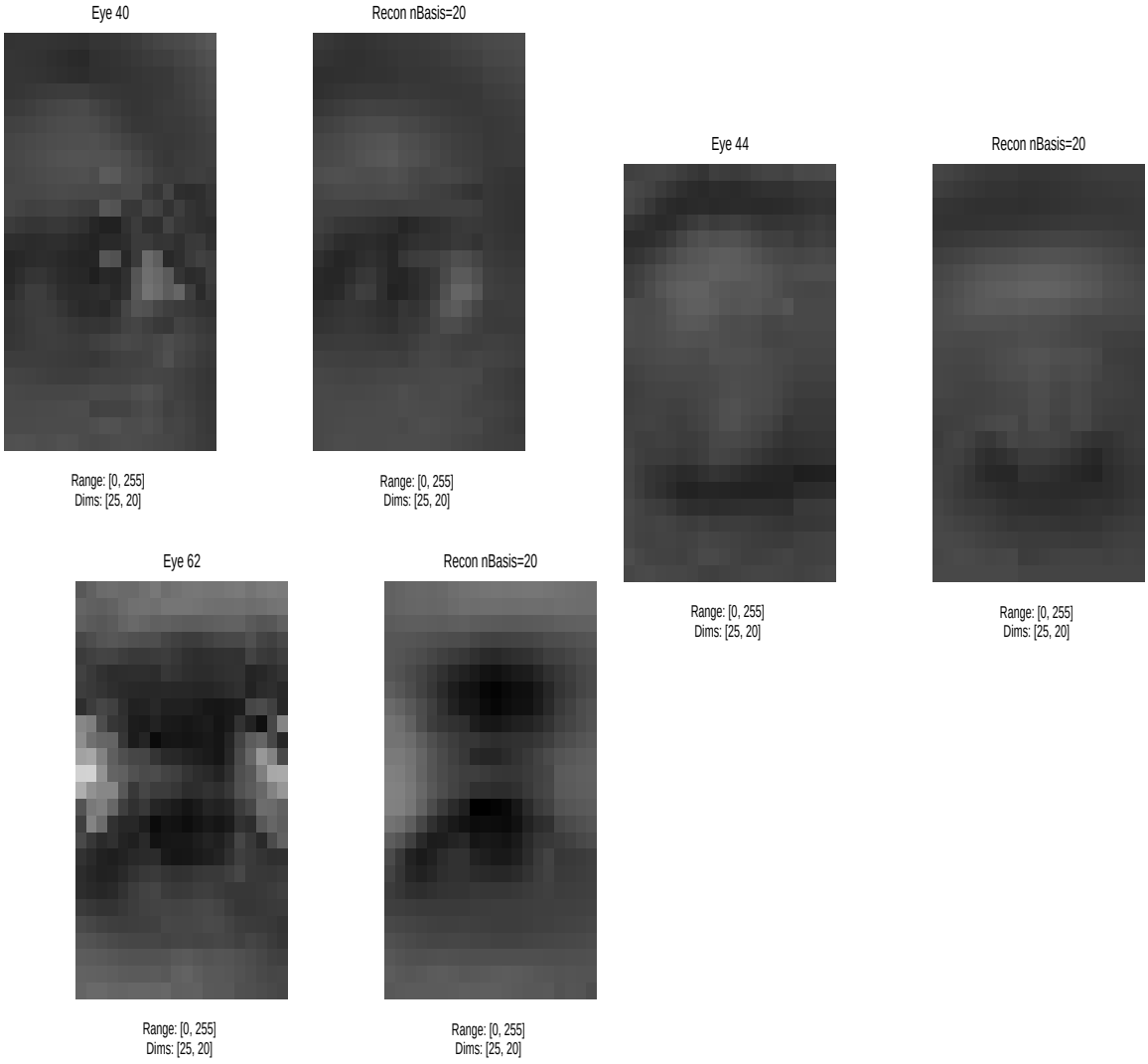
Another interesting plot shows the trade-off in true-detection versus true-rejection if we change the angle of the classification line in the out of subspace error plot.



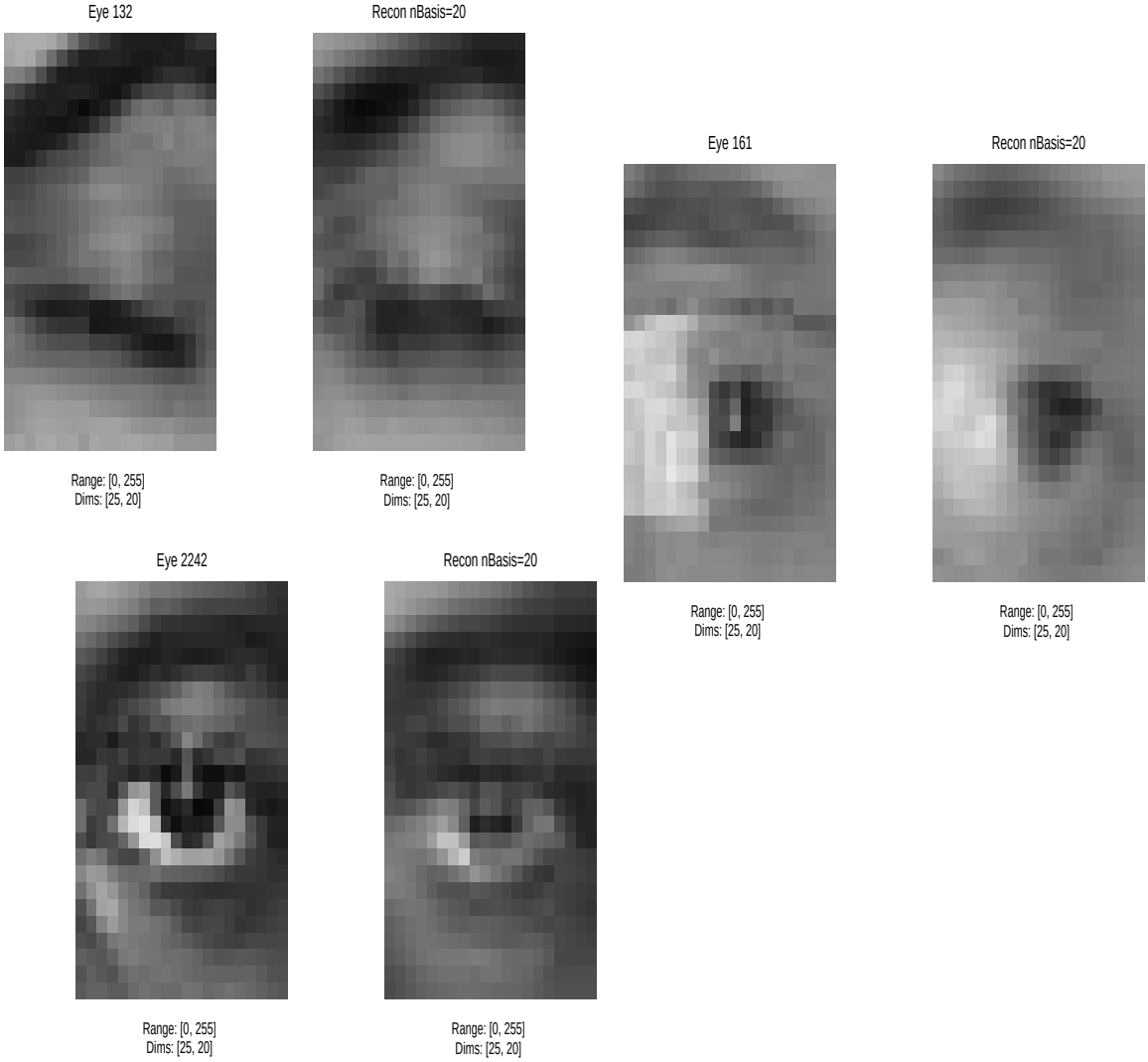
Basically, we can increase the rate of true-detection, but after a certain point, any increase comes at the expense of a decrease in the true-rejection rate.

Eigen Eyes

It is illustrative to look at some of the eye images that were wrongly classified as non-eyes



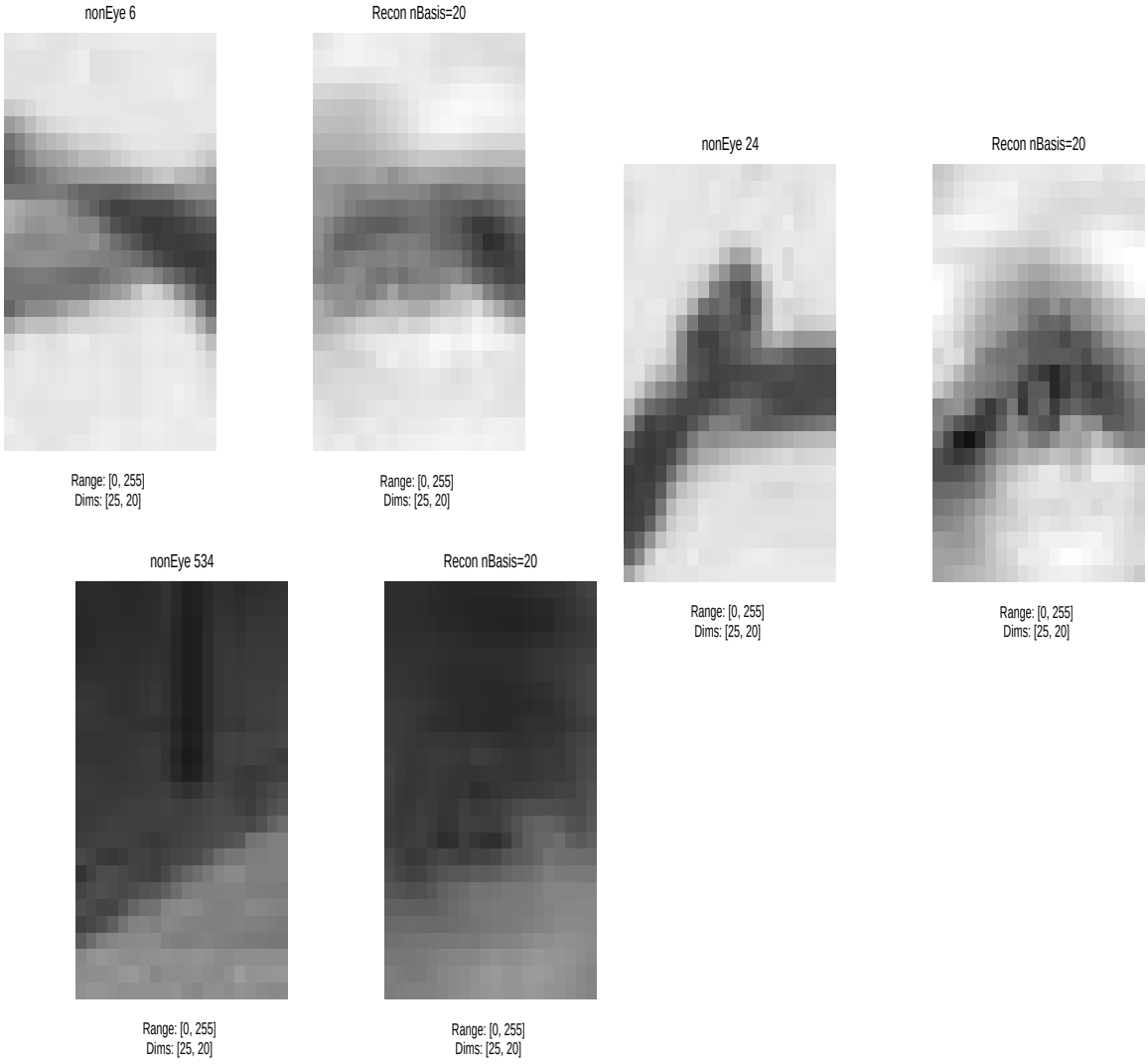
Eigen Eyes



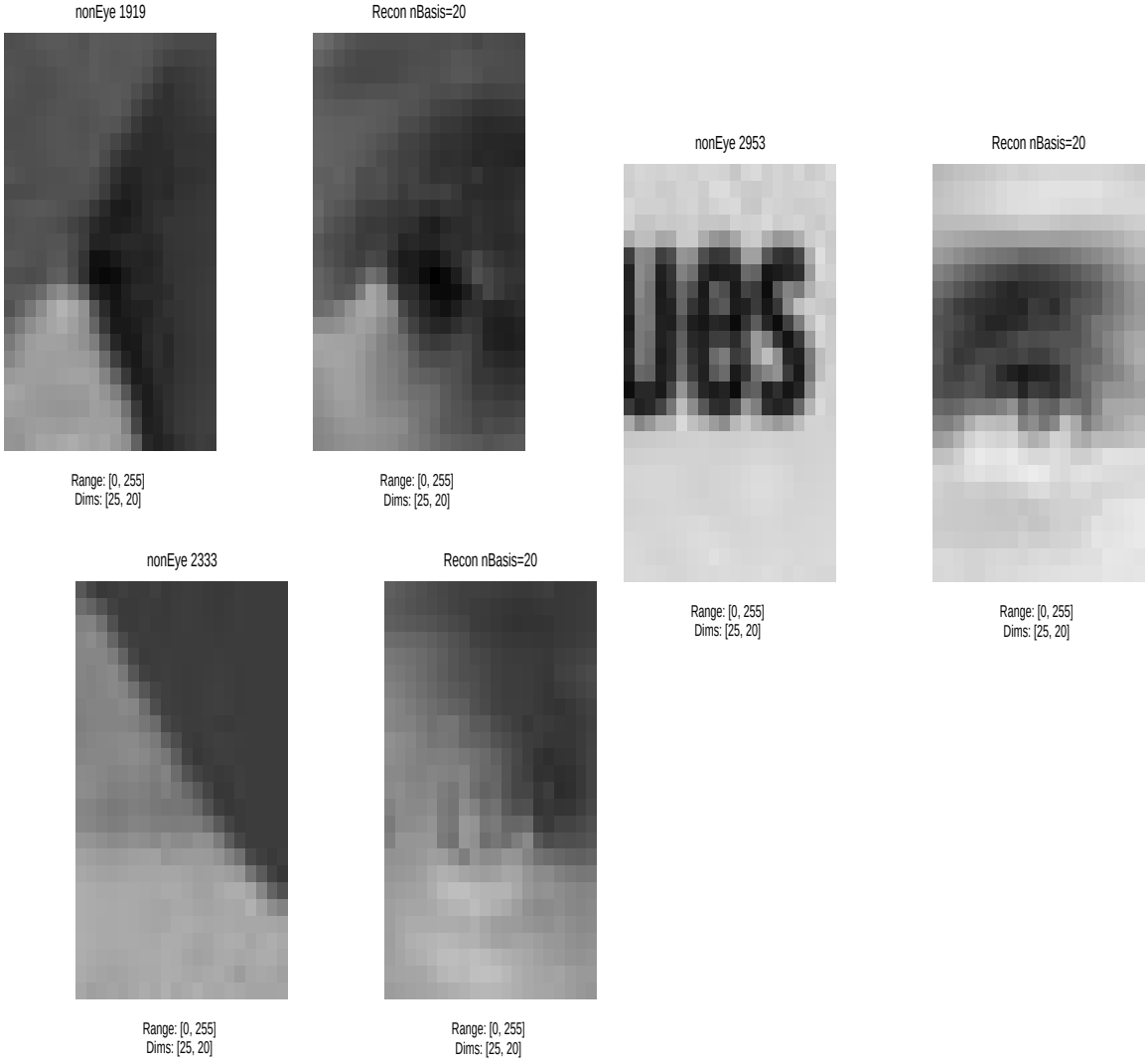
Many of these images have strange lighting artifacts, show closed eyes, or look very different to the average eye, however, some very reasonable eyes are also mis-classified.

Eigen Eyes

Now look at some of the non-eye images that were classified as eyes:



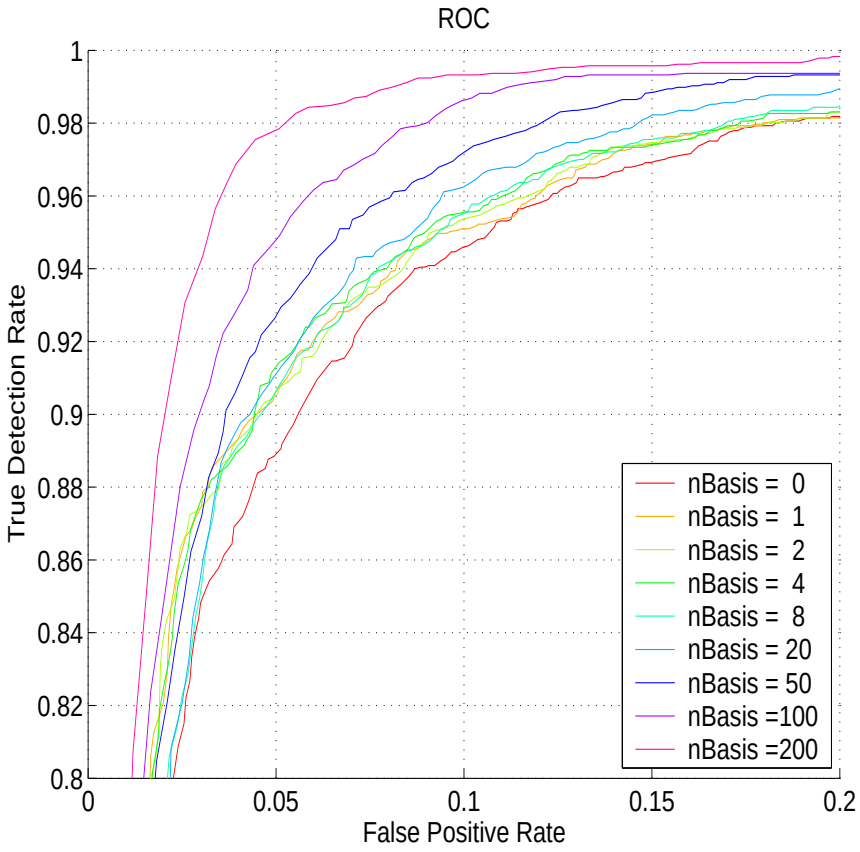
Eigen Eyes



Some have structure that roughly resembles the pattern of light and dark that we could expect in an eye, but others seem completely unrelated!

Eigen Eyes

There is a good reason for saving some of our data for testing, see what happens if we evaluate the classification performance of our eigen-basis on the data we used for training:



Notice that if we increase the number of basis functions used to represent images, we keep improving the true-detection vs false-positive ratio! however, all that we're seeing is over-fitting,

that is, the larger number of basis images we use, the better our eigen-basis becomes at explaining particularities of our training data that do not apply to the general domain of eye images.

Saving some of our data for testing, and for validation of our choice of eigen-basis, is a good idea. However it has a downside: If we have little training data to start with, and we save part of that for testing, we may find ourselves in a situation in which the remaining data is not enough for PCA (or in general, any other learning method) to capture interesting structure that is general to the domain we're analyzing.

Observations about PCA

PCA models data as a multi-dimensional ellipse, the PCA vectors give an orthogonal basis that describes the orientation of this ellipse:

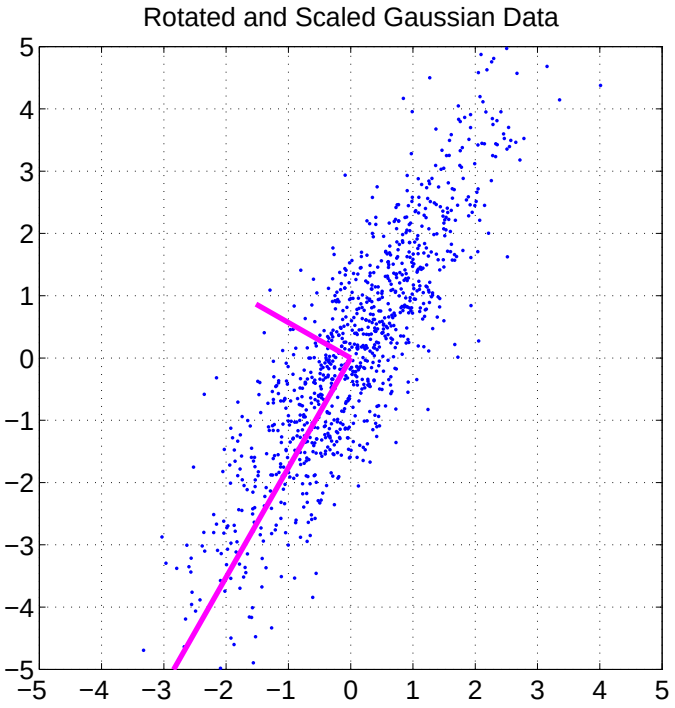


Figure 1: Gaussian data and principal directions from PCA

Which works well in many cases.

Observations about PCA

However, PCA does not distinguish a situation in which the observed data is not distributed as a multidimensional Gaussian:

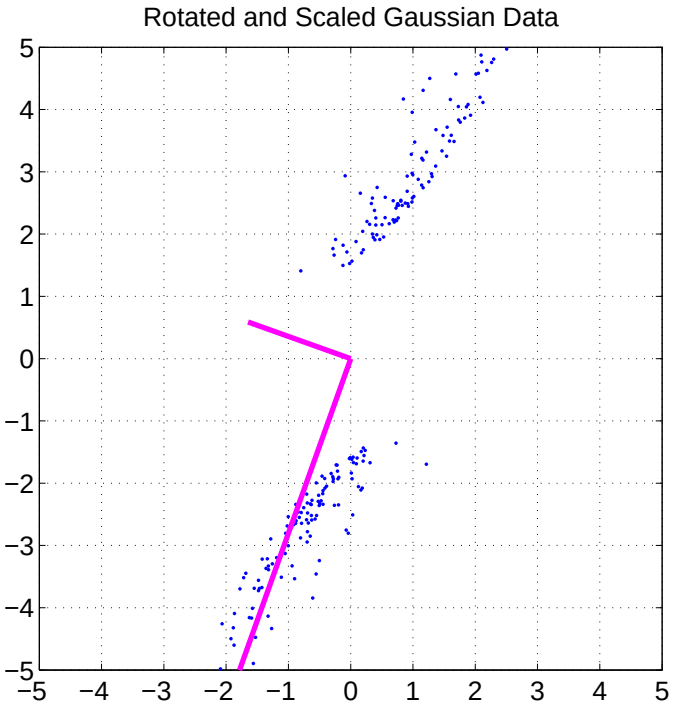


Figure 2: Data distributed along some manifold, and principal directions from PCA

PCA gives the vectors that best approximate the data as a multidimensional ellipse, if these data points correspond to a representative sample of some class we wish to model we have a problem...

Observations about PCA

If we try to do classification, we now have a situation in which data that is not part of the original manifold is well approximated by our PCA model:

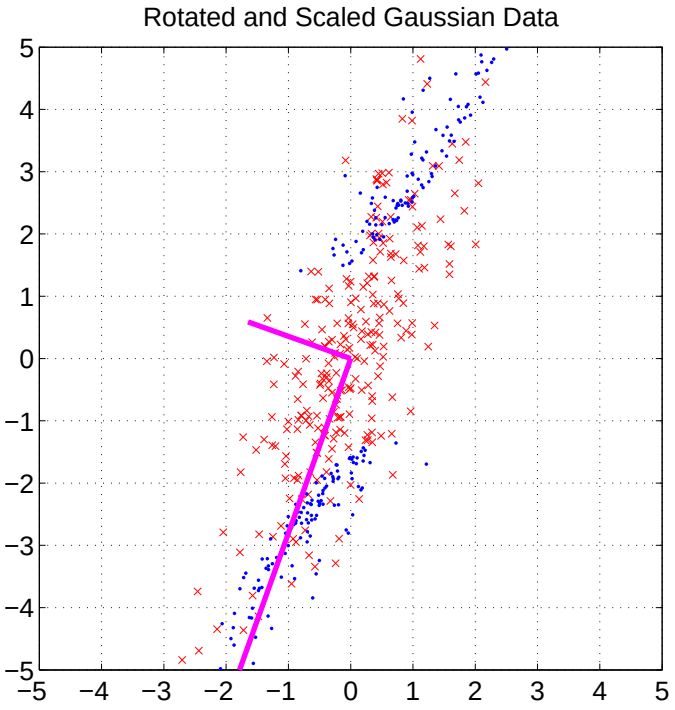


Figure 3: Red points correspond to data distributed according to our PCA model, blue points correspond to the original data.

If we use simple statistics based on reconstruction error and unexplained variance, we would end up incorrectly classifying many of the red points as instances of our class.

Recognition using parametric PCA

Murase and Nayar ('Visual Learning and Recognition of 3-D Objects from Appearance', IJCP, 14, 1995) propose that recognition can be carried out using PCA, together with parametric models of the manifolds described by the data from a particular object.

Given a set of P objects, and a set of images for each object that capture different poses and varying illumination, they build several PCA models:

- A universal model derived from the images of all the objects, under all pose and illumination conditions.
- P object specific models that include just the images of each particular object.

Recognition using parametric PCA

They show that the projection of the training images of a particular object onto these eigenspaces describes a manifold whose dimension corresponds to the number of parameters that describe the changes in pose and illumination.

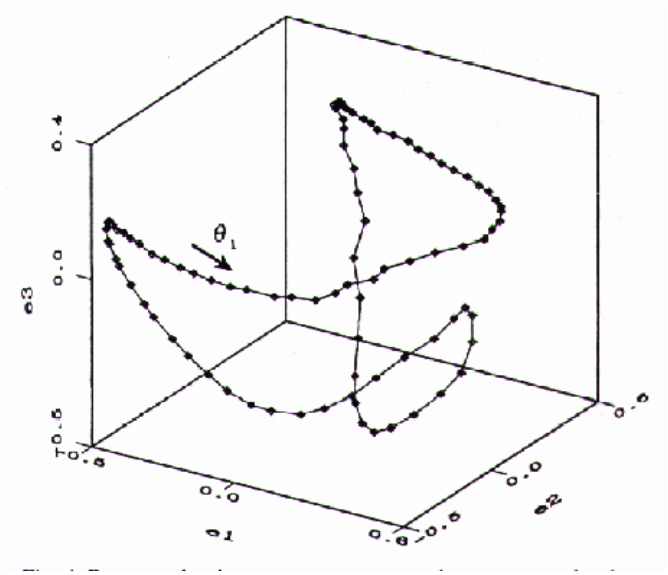


Figure 4: Manifold for images of a single object, with 1 parameter, using only the first 3 vectors of the eigenbasis (from Murase & Nayar, 1995).

Recognition using parametric PCA

Given a new image, classification consists of projecting the image onto the universal eigenspace, and finding the object manifold to which the projected image has the smallest distance.

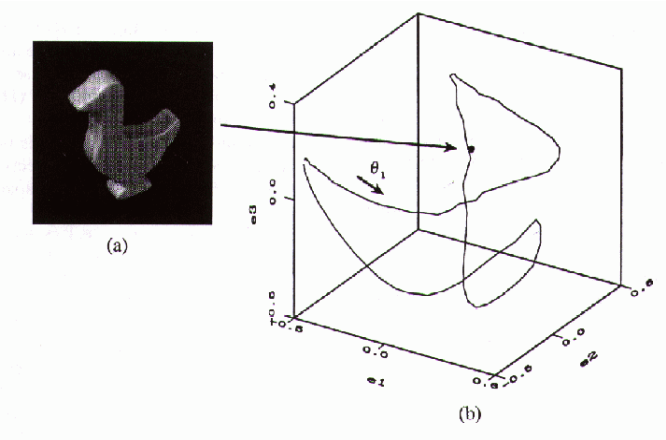


Figure 5: New image classification, projection onto the eigenspace followed by detection of the closest manifold (from Murase & Nayar 1995).

The pose is then estimated from the particular eigenspace of the corresponding object.

Recognition using parametric PCA

A few sample manifolds for different objects are shown below (in this case 2 parameters describe the changes in pose and illumination, so the manifolds represent a 2-d ribbon, shown here in the space corresponding to the 3 leading eigenvectors or each object-specific eigenspace).

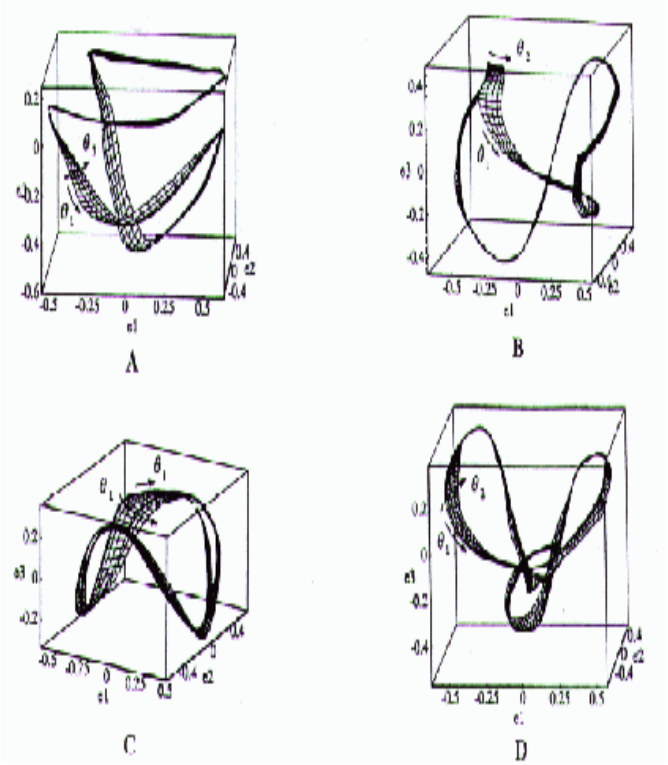


Figure 6: Representations of manifolds for different objects (from Murase & Nayar 1995).