Some more topics

In this class we have examined how digital data are manipulated, analyzed, and displayed by computer. Today we are going to briefly look at some more advanced processing topics to get an idea of other applications of these techniques.

Principal Component Analysis and Decorrelation Stretches

If we are examining false-color computer images, or at least images in which the actual colors are not required to match real-life colors, we can sometimes present much more detail by transforming the data such that the red, green, and blue channels contain data that are more independent of each other than the initial r, g, and b channels were. This has the effect of increasing the number of colors apparent in an image, making it easier to distinguish features. This approach has many applications in medicine and in remote sensing, aiding in the interpretation of images.

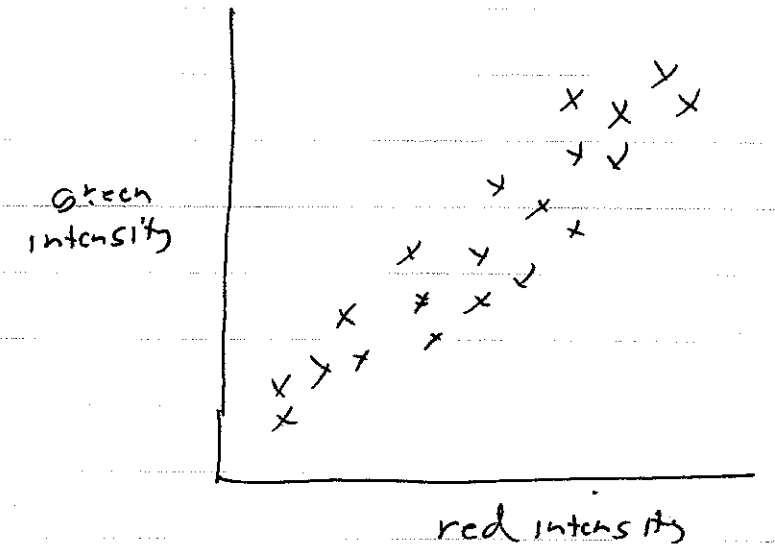
For example, consider an image in which the red, green, and blue images are 100% correlated with each other. This means that the r, g, and b channels ~~are related~~ differ from each other by at most a scale factor, such as

$$r(i,j) = A \cdot b(i,j) = B \cdot g(i,j)$$

In the case where $A = B = 1$, then this reduces to a black and white image.

Obviously you can distinguish more in a color image than in that image displayed only in black and white. Principal component analysis is a technique whereby we can transform images that are highly correlated with each other to images that are nearly independent of each other.

It is easier to picture what is going on here if we start with a two-color image rather than a three-color image. Pretend there are only two primary colors in the world, red and green. Now, create a scatterplot of pixels in an image by plotting each at the location defined by r and g for that pixel. You might get a plot that looks like this:



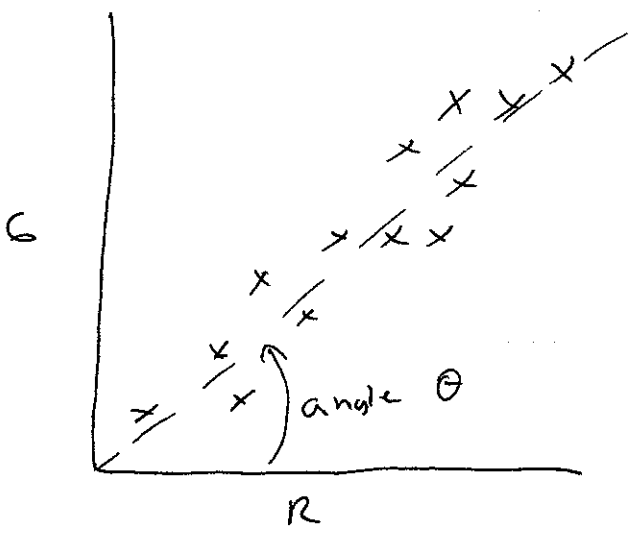
Note that while the observed pixels have a large range in red and a large range in green, a pixel that is bright in red tends to be bright in green as well.

This is what we mean by correlated images - the red and green values tend to track each other.

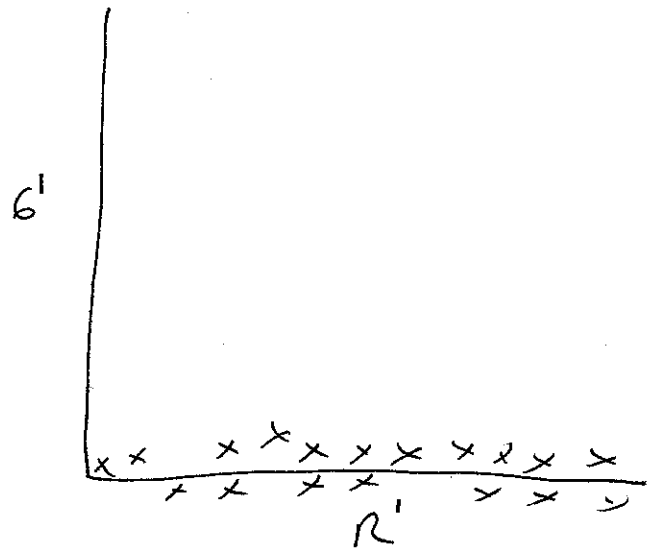
Such an image would tend to be mostly yellows of varying intensities, with some yellows a bit more greenish and others a bit more reddish.

How can we use more of the color space that is available? Since we have only two primary colors, our color space should include some pixels that are purely red, some purely green, and the assortment of yellows and ~~in~~ in-between colors we just mentioned.

Suppose we apply a rotation to the scatterplot we just made. We might end up with something like this:

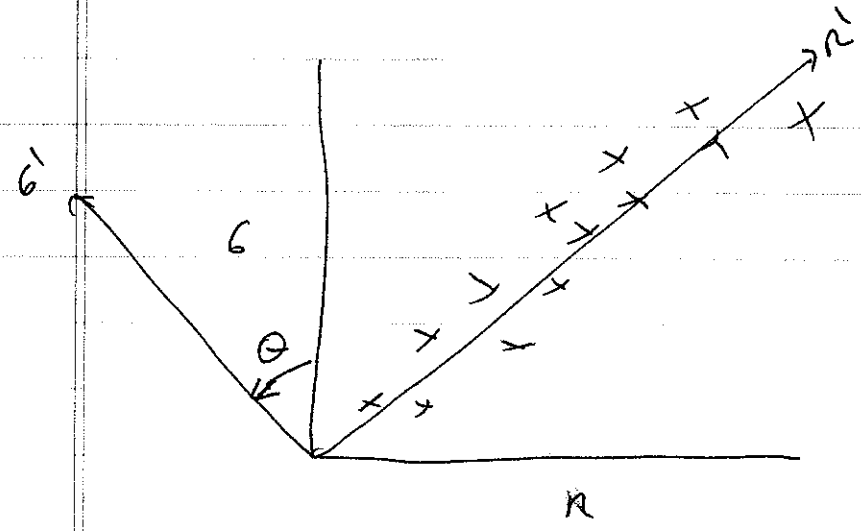


Input



Rotated ~~output~~ output through angle θ

Viewed another way, we could have defined a rotated coordinate system R', G' as



Note that we now observe the pixels to be well-distributed in R', and clumped near 0 in G'. R' is called the <u>first principal component</u> of the image, since it contains most of the variation of the points. G' is the <u>second principal component</u>, and has little variation. However, the variation in G' is not correlated with R', so it represents new information.

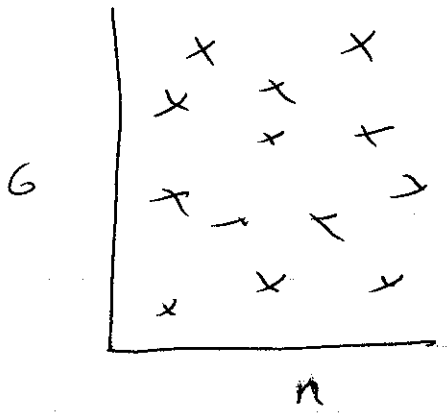
If we were to apply this transformation to each pixel in an image, we could obtain a new image (r', g') from an initial image (r, g) via the following equation:

$$\begin{pmatrix} r' \\ g' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} r \\ g \end{pmatrix}$$

and displaying (r', g') as a new image would have

points well spread out in r', but still rather clumped near 0 in g'. It may not be any easier to distinguish features here than in the original image.

But we know how to stretch and optimize channels to use up the available range in a display - we can simply scale g' to have a desired mean and standard deviation. This will increase the variability in g' and generate a better-distributed image. By this scaling, then, we would end up with a new scatter plot that looks like



Now points are well spread out in R and G, and are not correlated at all.

3-D version

The above worked well in 2-D images because we could easily visualize the rotation needed. How does this work for 3-color images? We will have to generalize the approach, and we do this with eigenvector decomposition of the correlation matrix.

Recall that what we want to do is derive a transformation that reduces correlated image channels to uncorrelated ones. Those of you who have have some communication theory have seen this before - we call this transformation the Karhunen-Loeve transformation in communication literature. In each case we need to start with the correlation properties of the image.

Correlation matrix. We start with the calculation of a covariance matrix, which gives the degree to which each channel is an image is relation to the others. The covariance function is defined as

$$C(i,j) = E\{(Col_i(x,y) - \mu_i)(Col_j(x,y) - \mu_j)\}$$

where Col_i is the ith color at location (x,y) and μ_i is the mean of that channel. Note that $C(i,j)$ reduces to

$$C(i,i) = E\{(Col_i(x,y) - \mu_i)(Col_i(x,y) - \mu_i)\}$$

or just the variance of that channel. We might numerically calculate $C(i,j)$ then as

$$C(i,j) = \frac{1}{N^2} \sum_{x=1}^{N} \sum_{y=1}^{N} (col_i(x,y) - \mu_i)(col_j(x,y) - \mu_j)$$

The covariance matrix $C(i,j)$ is a 3×3 matrix for a 3-color image, with i and j each becoming r, g, and b.

We next find the eigenvalues and eigenvectors of $C(i,j)$, where the eigenvector matrix A has the individual eigenvectors of C as its rows.

Using this new transformation, we can define a new image using

$$\begin{pmatrix} r' \\ g' \\ b' \end{pmatrix} = A \begin{pmatrix} r \\ g \\ b \end{pmatrix}$$

This is the generalization of the simple rotation we did previously into three channels.

Finally, we need to stretch each channel of the new matrix so that it fits into our 256-level display. This is the <u>decorrelation stretch</u> step to result in the final image.

<u>An example</u>.

Suppose we have an image with a covariance matrix that looks like

$$C(i,j) = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{pmatrix}$$

In general, we won't have zeros and round numbers, but this makes the math easier.

First, we need to find the eigenvalues of the matrix. We subtract the eigenvalue from each diagonal element and set the determinant to zero:

$$\begin{vmatrix} 2-\lambda & 0 & 1 \\ 0 & 2-\lambda & 0 \\ 1 & 0 & 2-\lambda \end{vmatrix} = 0$$

or

$$(2-\lambda)^3 - (2-\lambda) = 0$$

$$(2-\lambda)\left[(2-\lambda)^2 - 1\right] = 0$$

$$(2-\lambda)(1-\lambda)(3-\lambda) = 0$$

so $\lambda_1 = 3$, $\lambda_2 = 2$, and $\lambda_3 = 1$. We find each eigenvector by solving the matrix equation

$$AX = \lambda X$$

So, for $\lambda_1 = 3$:

$$\begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{pmatrix}\begin{pmatrix} r \\ g \\ b \end{pmatrix} = 3\begin{pmatrix} r \\ g \\ b \end{pmatrix}$$

$$\begin{aligned} 2r + b &= 3r \\ 2g &= 3g \\ r + 2b &= 3b \end{aligned} \quad \Rightarrow \quad r = b, g = 0 \quad \text{so} \quad U_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$\begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{pmatrix}\begin{pmatrix} r \\ g \\ b \end{pmatrix} = 2\begin{pmatrix} r \\ g \\ b \end{pmatrix} \quad \begin{matrix} 2r+b = 2r \\ 2g = 2g \\ r+2b = 2b \end{matrix} \Rightarrow v_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

and also we'll find $v_3 = \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$

The transformation matrix A is formed by making its rows the eigenvectors:

$$A = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix}$$

So, if we start with a pixel with $(r, g, b) = (5, 3, 4)$, we'll get

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix}\begin{pmatrix} 5 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} \frac{9}{\sqrt{2}} \\ 3 \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

for the new r, g, and b values. We do this to every pixel in the image, and apply the stretch independently to each channel to get a color-diverse image.