IMAGE COMPRESSION WITH THE SINGULAR VALUE DECOMPOSITION

Working with an image in R

Let's take an image of a leader that we all know and respect:

This image can be downloaded from the IAA website, after clicking on the link under People > Founding Director. It is also located on the course webpage at `http://birch.iaa.ncsu.edu/~slrace/LinearAlgebra2021/index.html`

Let's read this image into R. You'll need to install the pixmap package:

```
> #install.packages("pixmap",repos="http://R-Forge.R-project.org")
> library(pixmap)
```

Download the image to your computer and then set your working directory in R as the same place you have saved the image:

```
> setwd("FILEPATH TO PICTURE")
```

The first thing we need to do is convert the image into either a [R,G,B] (extension .ppm) or a grayscale (extension .pgm). Let's start with the [R,G,B] image and see what the data looks like in R:

```
> #system("convert rappa.jpg rappa.ppm")
> rappa = read.pnm("rappa.ppm")
> #Show the type of the information contained in our data:
> str(rappa)
```

```
Formal class 'pixmapRGB' [package "pixmap"] with 8 slots
  ..@ red     : num [1:160, 1:250] 1 1 1 1 1 1 1 1 1 1 ...
  ..@ green   : num [1:160, 1:250] 1 1 1 1 1 1 1 1 1 1 ...
  ..@ blue    : num [1:160, 1:250] 1 1 1 1 1 1 1 1 1 1 ...
  ..@ channels: chr [1:3] "red" "green" "blue"
  ..@ size    : int [1:2] 160 250
  ..@ cellres : num [1:2] 1 1
  ..@ bbox    : num [1:4] 0 0 250 160
  ..@ bbcent  : logi FALSE
```

You can see we have 3 matrices included in the pixmapRGB object in our global environment - one for each of the colors: red, green, and blue. Each of these matrices is 160×250 - that is the dimensions of the image in pixels. Each entry (i, j) in each matrix represents a single pixel, and the value in that location represents the intensity of each color in each pixel.

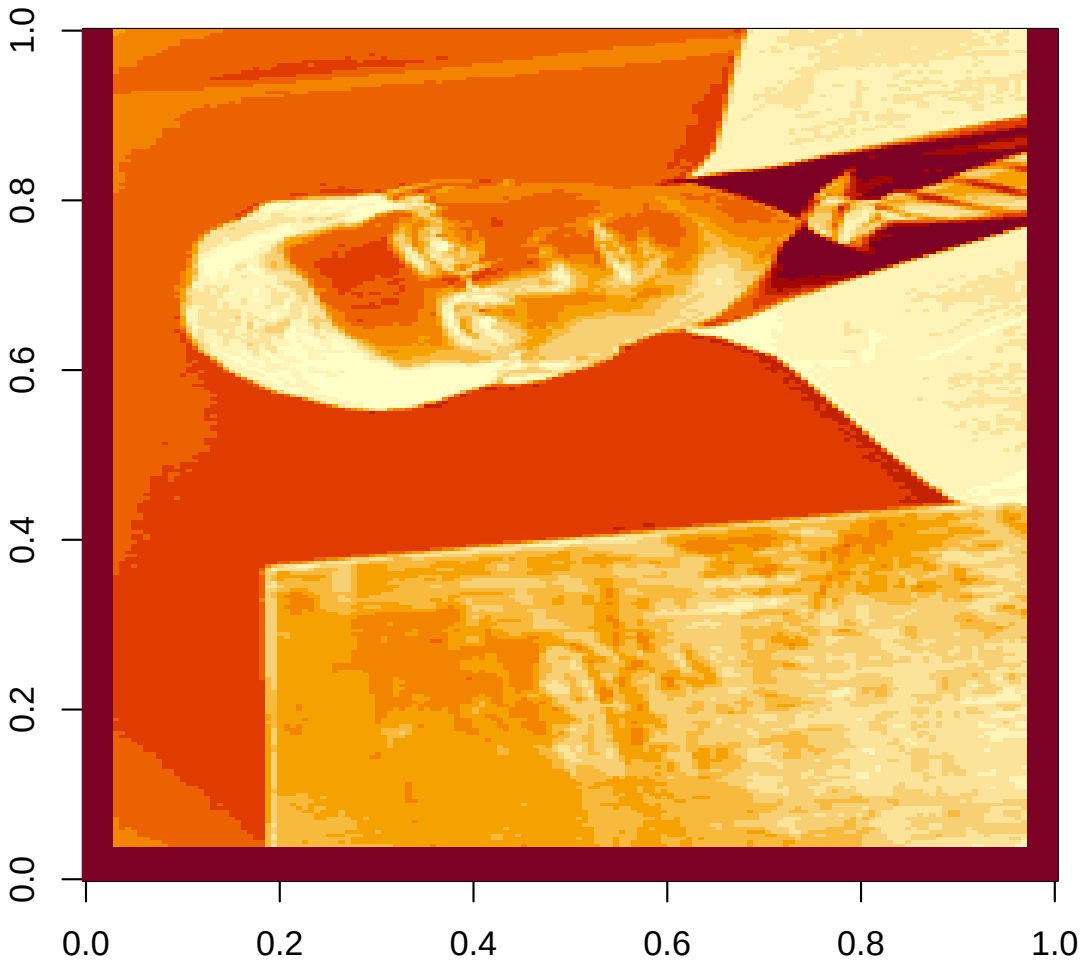
The pixmapRGB object is not a traditional data frame, and we have to reference the matrices within this object with '@' rather than with '$'

```
> rappa@size
```

```
[1] 160 250
```

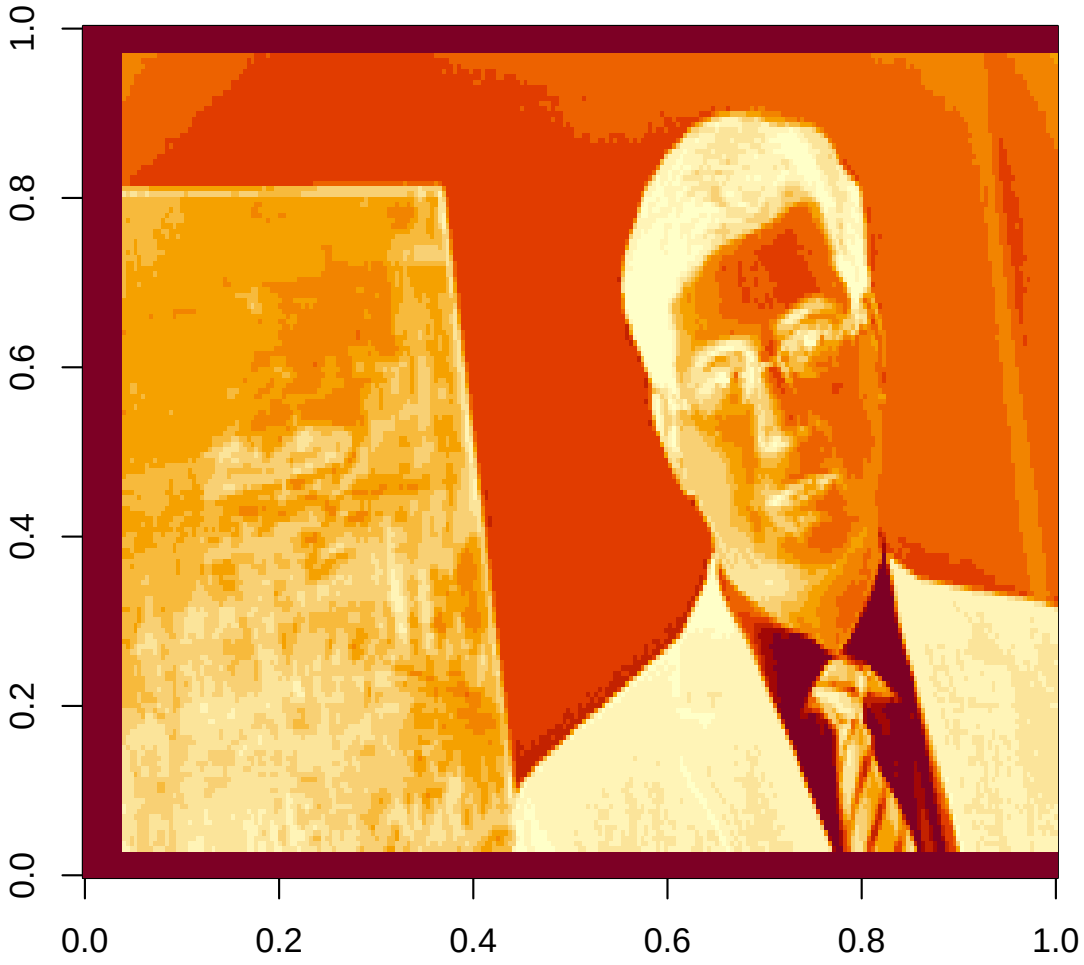
We can then display a heat map showing the intensity of each individual color in each pixel:

```
> rappa.red=rappa@red
> rappa.green=rappa@green
> rappa.blue=rappa@blue
> image(rappa.green)
```



Oops! He is sideways. To rotate the graphic, we actually have to rotate our coordinate system. There is an easy way to do this (with a little bit of matrix experience), we simply transpose the matrix and then reorder the columns so the last one is first. Note that nrow(rappa.green) in the code below refers to the number of columns in the transposed matrix.

```
> rappa.green=t(rappa.green)[,nrow(rappa.green):1]
> image(rappa.green)
```



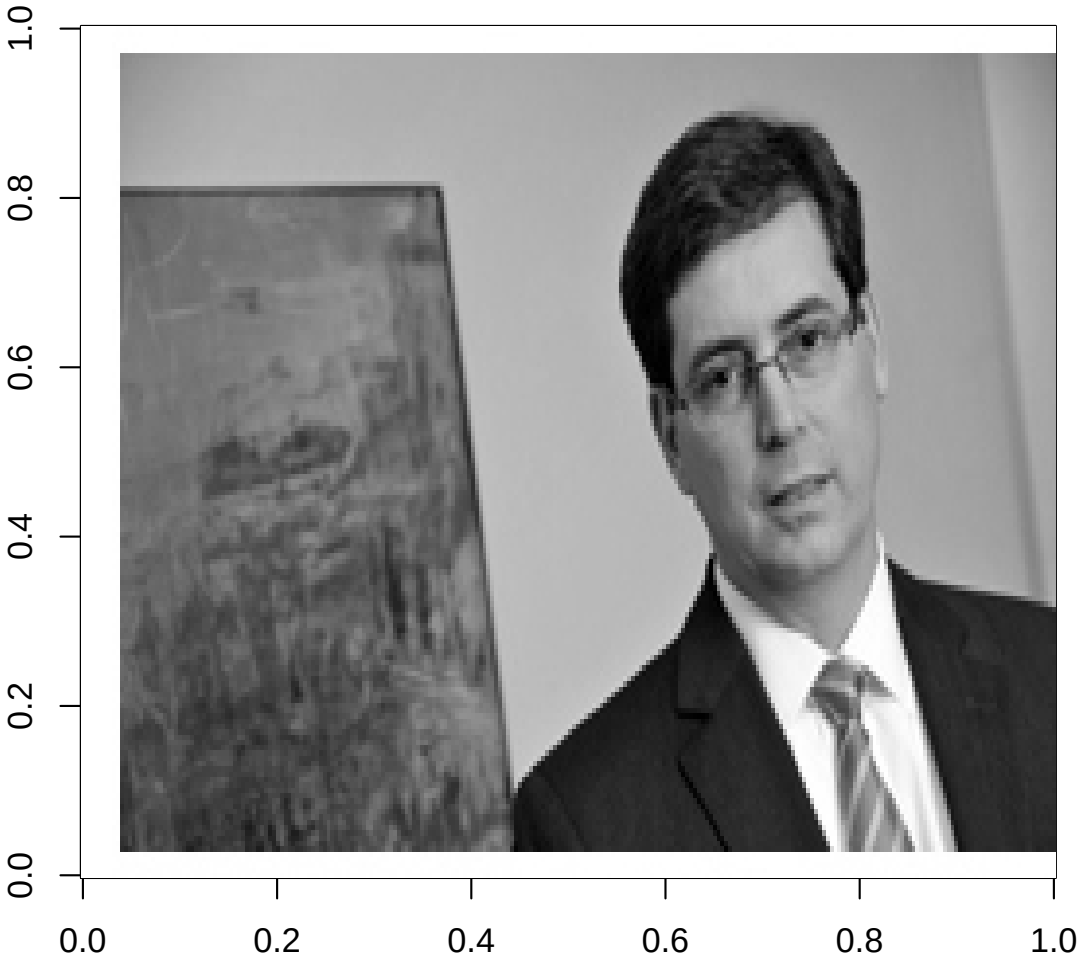
Rather than compressing the colors individually, let's work with the grayscale image:

```
> #system("convert rappa.jpg rappa.pgm")
> greyrappa = read.pnm("rappa.pgm")
> str(greyrappa)
```

```
Formal class 'pixmapGrey' [package "pixmap"] with 6 slots
  ..@ grey    : num [1:160, 1:250] 1 1 1 1 1 1 1 1 1 1 ...
  ..@ channels: chr "grey"
  ..@ size    : int [1:2] 160 250
  ..@ cellres : num [1:2] 1 1
  ..@ bbox    : num [1:4] 0 0 250 160
  ..@ bbcent  : logi FALSE
```

```
> rappa.grey=greyrappa@grey
> #again, rotate 90 degrees
> rappa.grey=t(rappa.grey)[,nrow(rappa.grey):1]


> image(rappa.grey, col=grey((0:1000)/1000))
```


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

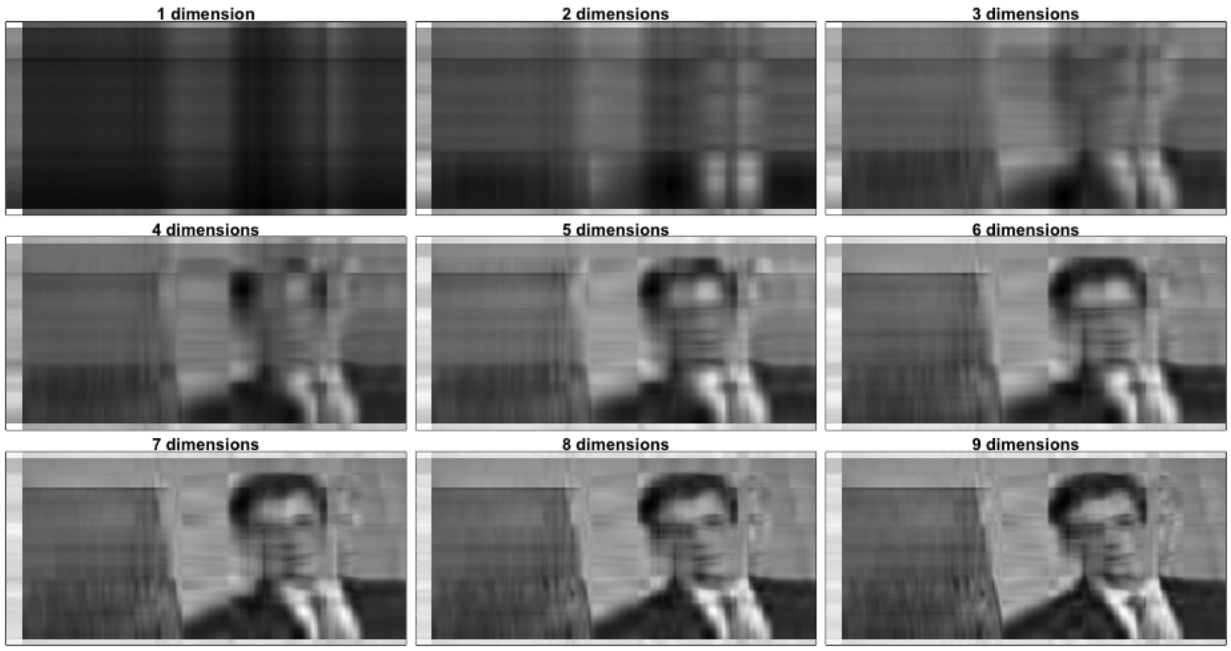
The Rank-Reduced Image

Now, let's use what we know about the SVD to compress this image into a lower dimensional space. First, let's compute the SVD and save the individual components. Remember that the rows of $\mathbf{V}^T$ are the right singular vectors. R outputs the matrix $\mathbf{V}$ which has the singular vectors in columns.

```
> rappasvd=svd(rappa.grey)
> U=rappasvd$u
> d=rappasvd$d
> Vt=t(rappasvd$v)
```

Now let's compute some approximations of this data matrix using different numbers of components. Let's start with just the first 9 components, and observe the changes to the image as we add each of these components:

```
> par(mfrow=c(3,3), mar=c(0.3,0.3,1,0.3))
> RappaRank_1 =  d[1]*U[,1] %o% Vt[1,] # stupid outer product function required here
> image(RappaRank_1,
+       col=grey((0:1000)/1000),
+       main=paste("1 dimension"),
+       xaxt = 'n',
+       yaxt = 'n')
> ind=2:9
> for (k in ind){
+   RappaRank_k =  U[,1:k] %*% diag(d[1:k]) %*% Vt[1:k,]
+   image(RappaRank_k,
+         col=grey((0:1000)/1000),
+         main=paste(k,"dimensions"),
+         xaxt = 'n',
+         yaxt = 'n')
+ }
```



How many singular vectors does it take to recognize Dr. Rappa? Nine is almost sufficient for those who know him best! Let's show the image progression from 10 to 90 components, adding 10 components each time.

```
> par(mfrow=c(3,3), mar=c(0.3,0.3,1,0.3))
> ind=seq(10,90,by=10)
> for (k in ind){
+   RappaRank_k =  U[,1:k] %*% diag(d[1:k]) %*% Vt[1:k,]
+   image(RappaRank_k,
+         col=grey((0:1000)/1000),
+         main=paste(k,"dimensions"),
+         xaxt = 'n',
+         yaxt = 'n')
+ }
```

By the time we get to 30 components, there is really no argument. Dr. Rappa is recognizable.

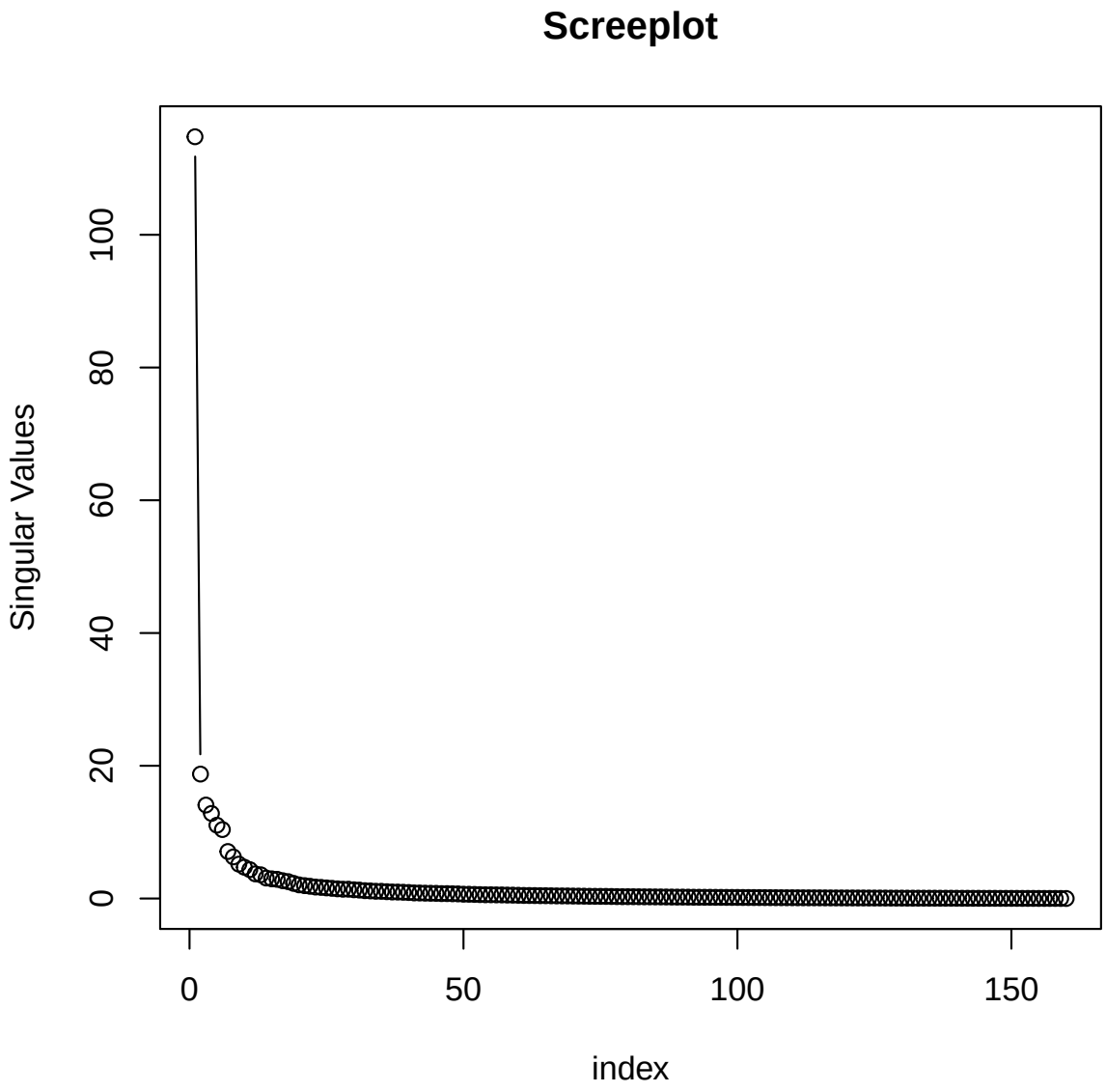
The Noise

One of the main benefits of the SVD is that the *signal-to-noise* ratio of each component decreases as we move towards the right end of the SVD sum. If $\mathbf{X}$ is our data matrix (in this example, it is a matrix of pixel data to create an image) then,

$$\mathbf{X} = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \sigma_3 \mathbf{u}_3 \mathbf{v}_3^T + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T$$

where r is the rank of the matrix. Our image matrix is full rank, $r = 160$. This is the number of nonzero singular values, σ_i. But, upon examinination of the screeplot, we see many of the singular values are nearly 0.

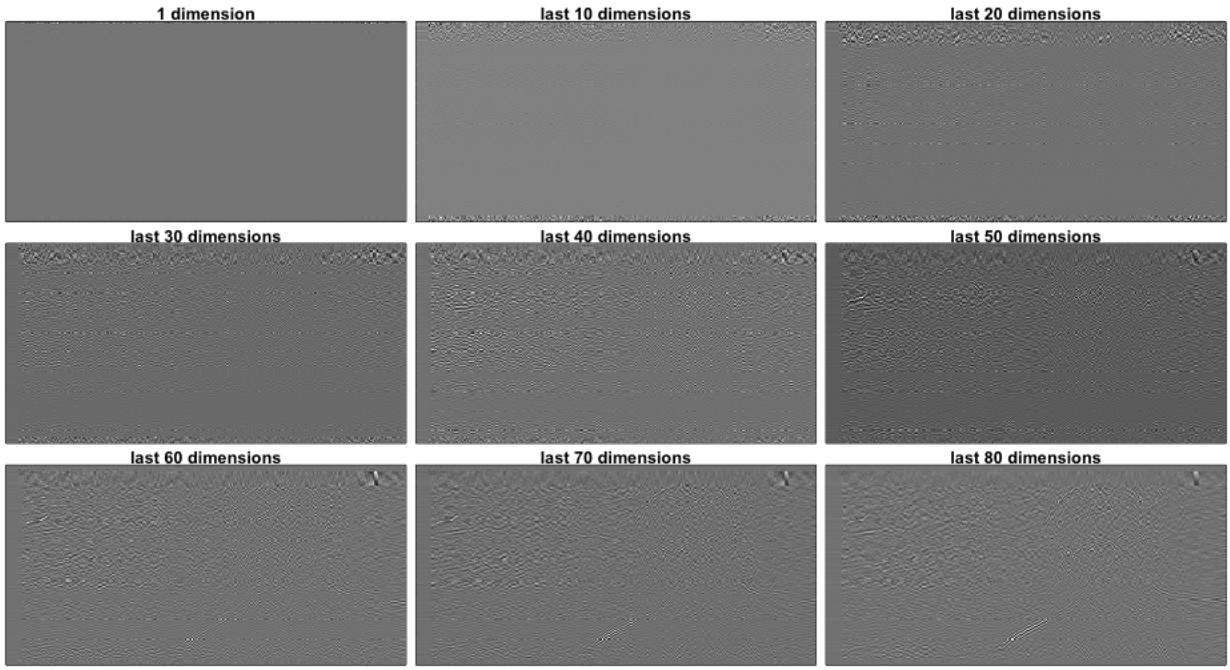
```
> plot(d,type='b', main='Screeplot',xlab='index', ylab = 'Singular Values')
```



We can think of these values as the amount of "information" directed along the singular components. If we assume the noise in the image or data is uniformly distributed along each orthogonal component $\mathbf{u}_i \mathbf{v}_i^T$, then there is just as much noise in the component $\sigma_1 \mathbf{u}_1 \mathbf{v}_1^T$ as there is in the component $\sigma_{160} \mathbf{u}_{160} \mathbf{v}_{160}^T$. But, as we've just seen, there is far less information in the component $\sigma_{160} \mathbf{u}_{160} \mathbf{v}_{160}^T$ than there is in the component $\sigma_1 \mathbf{u}_1 \mathbf{v}_1^T$. This means that the later components are primarily noise. Let's see if we can illustrate this using our image. We'll construct the parts of the image that are represented on the *last* singular components. We'll start with the last 10 components and progress up to the last 90 components.

```
> par(mfrow=c(3,3), mar=c(0.3,0.3,1,0.3))
> RappaRank_160 =  d[160]*U[,160] %o% Vt[160,] # stupid outer product function required here
> image(RappaRank_160,
```

```
+       col=grey((0:1000)/1000),
+       main=paste("1 dimension"),
+       xaxt = 'n',
+       yaxt = 'n')
> ind=seq(150,80,by=-10)
> for (n in ind){
+   RappaRank_n =  U[,n:160] %*% diag(d[n:160]) %*% Vt[n:160,]
+   image(RappaRank_n,
+         col=grey((0:1000)/1000),
+         main=paste("last", (160-n),"dimensions"),
+         xaxt = 'n',
+         yaxt = 'n')
+ }
```



It's noise! In the last 80 dimensions, we can't even tell there is a human in the picture! Let's continue on, just for fun:

```
> par(mfrow=c(2,3), mar=c(0.3,0.3,1,0.3))
> ind=seq(70,20,by=-10)
> for (n in ind){
+   RappaRank_n =  U[,n:160] %*% diag(d[n:160]) %*% Vt[n:160,]
+   image(RappaRank_n,
+         col=grey((0:1000)/1000),
+         main=paste("last", (160-n),"dimensions"),
+         xaxt = 'n',
+         yaxt = 'n')
+ }
```

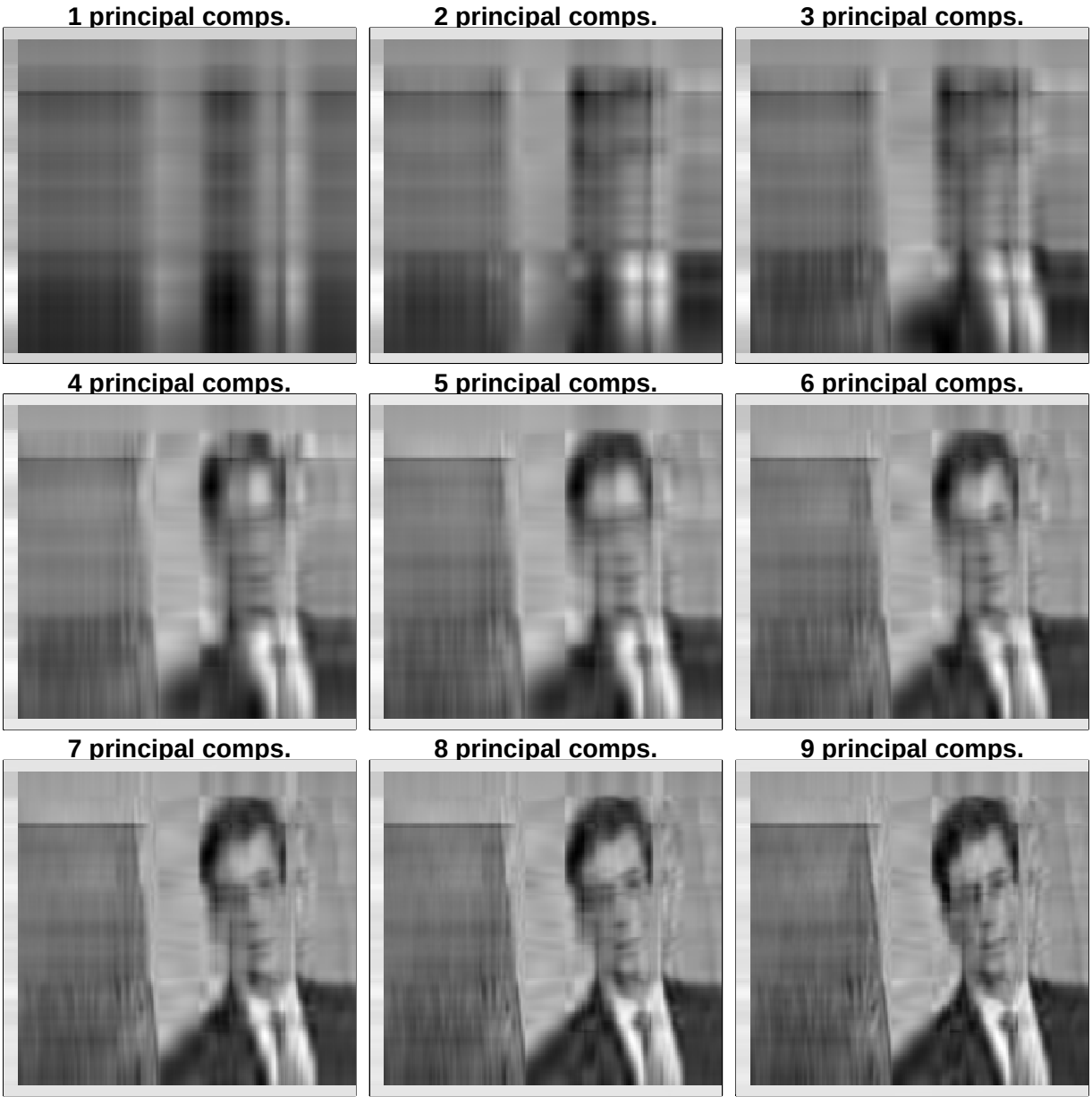
Finally, as we reach the last $\sim$ 90-100 components, we see the outline of Dr. Rappa. One of the first things to go when images are compressed are the crisp outlines of objects. This is something you may have witnessed in your own experience, particularly when changing the format of a picture to one that compresses the size.

What happens when we use PCA instead?

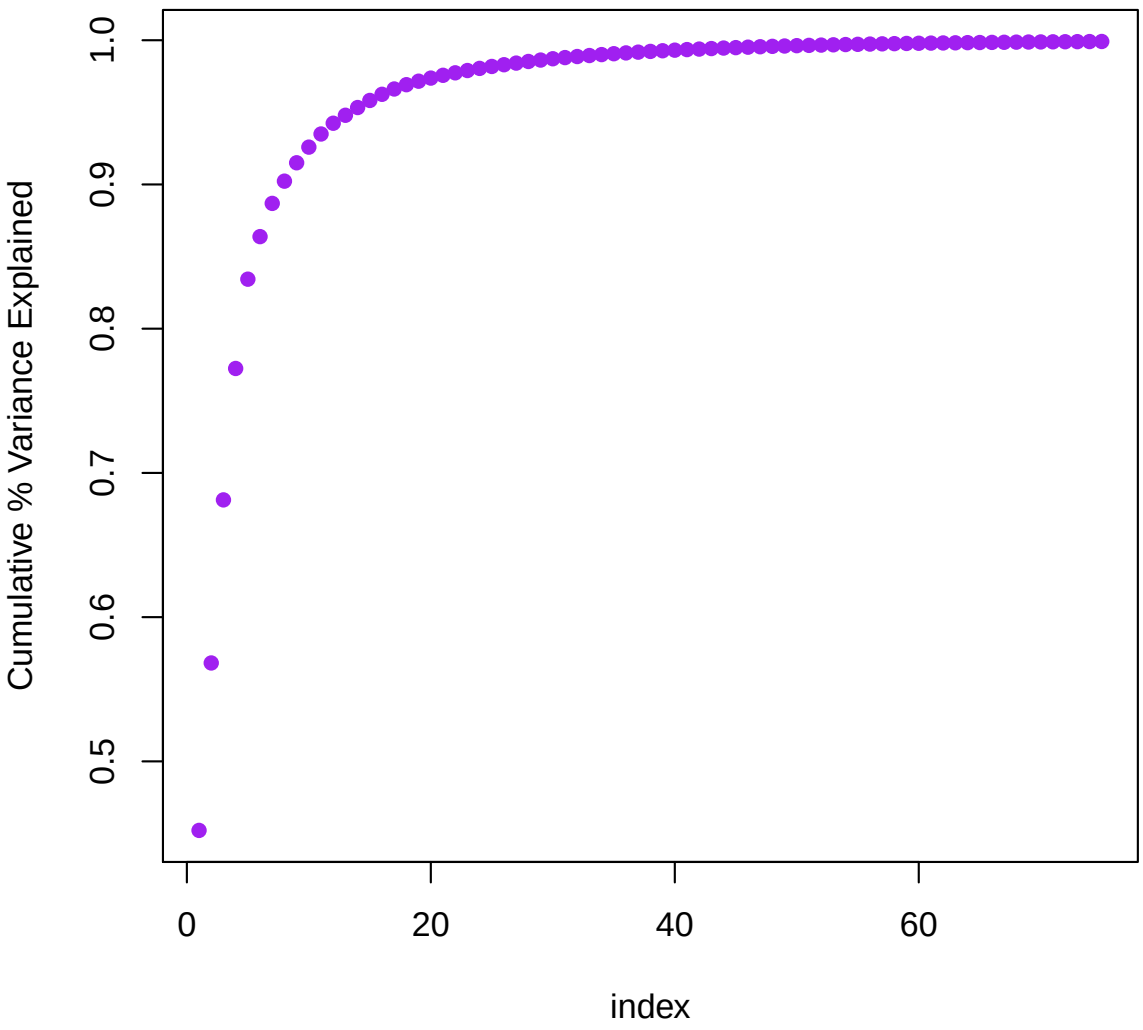
The result is not much different! The one advantage this might have educationally is that you can relate what you're seeing in the images to proportion of variance explained. The one disadvantage is that to recreate the image as we expect to see it, we have to add the mean back in to the image because the centering actually destroys the *look* of the image. In the following code, I'll show the same progression along the first 90 components - this time with *principal*

components rather than *singular components*. I will also show a plot of the cumulative variance explained so you can see how each picture matches each % of variance explained.

```
> par(mfrow=c(3,3), mar=c(0.3,0.3,1,0.3))
> pca=prcomp(rappa.grey)
> u2=pca$rotation
> dvt2=t(pca$x)
> centroid = colMeans(rappa.grey)
> ind=1:9
> for (k in ind){
+   if (k==1) {pcak =  t(u2[,1]%o%dvt2[1,])+rep(1,250)%o%centroid}
+   else {pcak =  t(u2[,1:k] %*% dvt2[1:k,])+rep(1,250)%o%centroid}
+   image(pcak,
+         col=grey((0:1000)/1000),
+         main=paste(k,"principal comps."),
+         xaxt = 'n',
+         yaxt = 'n')}
```




```
> plot(cumsum(pca$sdev[1:75]^2)/sum(pca$sdev^2), ylab = 'Cumulative % Variance Explained',col='purple',pch=16, xlab='index
```



What's the *point*?

The main purpose of this example was to reassure you that even using a small number of components can give you a nearly complete view of it. Because of the noise that necessarily comes with measurement of large amounts of correlated information, we don't lose a tremendous amount of *signal* when we reduce dimensionality.

I hope that students will keep the image of 5-dimensional Dr. Rappa in their heads when they consider what they are losing in the projection onto a subset of principal components. The important part of the picture resolves first, followed by the finer edges and details.