cryptoPunk Prinicipal Components

Building the matrix

First step: flatten the images into arrays of 1728 values between 0 and 255… Then, take the mean for each value in each observation to produce the average image by pixel. Big matrix. Not huge.

```r
pngs = list.files(path = "./originals", pattern = ".png")

T = matrix(0, 1728, length(pngs))

for (j in 1:length(pngs)){
  img = readPNG(paste("./originals/", pngs[j], sep=''))
  T[,j] = c(as.vector(img[,,1]), as.vector(img[,,2]), as.vector(img[,,3]))
}

dim(T)
```

```
## [1]  1728 10000
```

```r
avgPic = rowMeans(T)
T = T - avgPic
```

Test plots

Plot the first cryptoPunk with no transformations.

```r
j = readPNG(paste("./originals/", pngs[1], sep=''))

par(pty="s")
plot(1:2, type='n', xlab='', ylab='', axes=F,
     main=pngs[1])

rasterImage(j, 1, 1, 2, 2, interpolate = FALSE)
```

0000.png

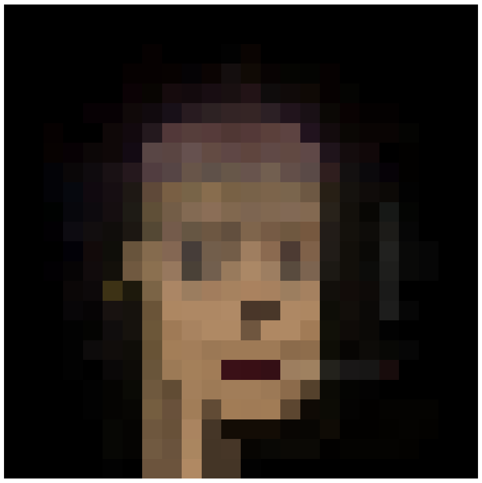
Plot the average cryptoPunk.

```
meanPic = array(avgPic, dim=c(24,24,3))

par(pty="s")
plot(1:2, type='n', xlab='', ylab='', axes=F,
     main="Average CryptoPunk")

rasterImage(meanPic, 1, 1, 2, 2, interpolate = FALSE)
```

Average CryptoPunk



Subtract the initial image from the average, plot the new image. This is a representation of all the things that are unique about it, different from the mean.

```
s1_mean_centered = array(pmax(T[,1], 0),
                         dim = c(24, 24, 3))

par(pty="s")
plot(1:2, type = 'n', xlab = '', ylab = '', axes = F,
     main =  '')

rasterImage(s1_mean_centered, 1, 1, 2, 2, interpolate = FALSE)
```

Singular Value Decomposition

We min/max scale the values representing each pixel so that they are always with the range of 256. This function will plot a principal component as given by singular value decomposition, given the index of the desired component, a matrix U, and the proper image dimensions.

```r
plotEigenpunk <- function(n, U = Tsvd$u, dim=c(24, 24, 3)) {
  u_range = max(U[,n]) - min(U[,n])
  scaled_u = U[,n] / u_range
  scaled_u = scaled_u - min(scaled_u)
  scaled_u = array(scaled_u, dim=dim)

  # This step here fixes a bug where color intensity of 1 is outside
  # the open interval [0,1]
  scaled_u[scaled_u > 0.9999999] <- 0.9999990

  plot(1:2, main = n, type='n', xlab='', ylab='', axes=F)
  rasterImage(scaled_u, 1, 1, 2, 2, interpolate = FALSE)
}
```

We calculate the singular value decomposition and plot an EigenPunk. This is the first eigenvector of the matrix of 10k images, which accounts for more variance of the entire set than any other component.

We may say that this image is a significant part of more original cryptoPunks than any other. It might be realistic to say that this contains more information about the set than the average does.

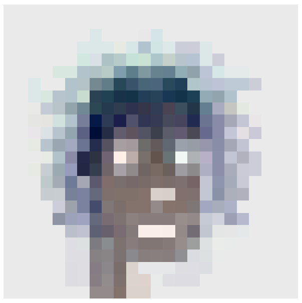
```
Tsvd = svd(T)
```

As we go down the gradient of eigenvectors, the components become less and less descriptive of the set… Still they all contain real information.

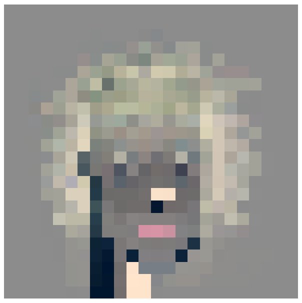
```
par(pty="s", mfrow = c(2,2), cex=0.7, mai=c(0.3,0.3,0.3,0.3))

plotEigenpunk(1)
plotEigenpunk(2)
plotEigenpunk(3)
plotEigenpunk(4)
```

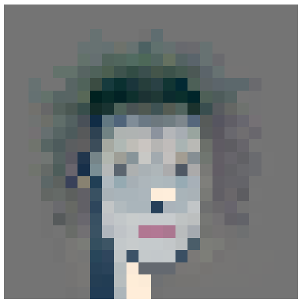
1



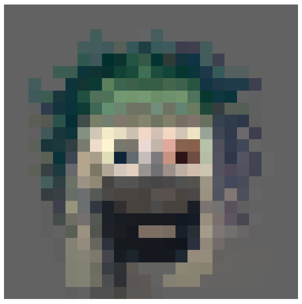
2



3



4

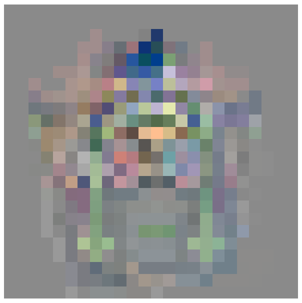


```
plotEigenpunk(7)
plotEigenpunk(10)
plotEigenpunk(50)
plotEigenpunk(200)
```

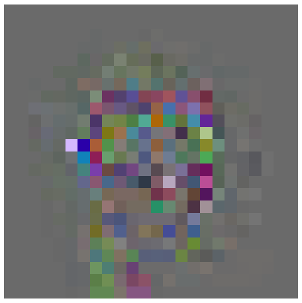
7

10

50



200



Now let's write the images.

```r
writeEigenPunk <- function(n, U, dim=c(24, 24, 3)) {
  png(paste0(n, '.png'))
  plotEigenpunk(n, U, dim)
  dev.off()
}

for (i in 1:1728) {
  writeEigenPunk(i, Tsvd$u)
}
```

We can compute how significant each of these images is in terms of the variance among the entire set of originals.

```r
vars <- Tsvd$d[1:200] / sum(Tsvd$d)
head(vars)
```

```
## [1] 0.05318563 0.04485044 0.04140603 0.02702552 0.02557952 0.02448105
```

Then, of course we can see cumulatively how many components are necessary to account for say 95% of the variance. . . . we need 194 components for this:

```r
(cumulative.var <- cumsum(Tsvd$d[1:200] / sum(Tsvd$d)))[1:194]
```

```
##   [1] 0.05318563 0.09803607 0.13944210 0.16646762 0.19204714 0.21652820
##   [7] 0.23949887 0.26122575 0.28106332 0.29958897 0.31727312 0.33375764
##  [13] 0.34831576 0.36271951 0.37624805 0.38917176 0.40185099 0.41402370
##  [19] 0.42595566 0.43738027 0.44853635 0.45895310 0.46911809 0.47895750
##  [25] 0.48859785 0.49798726 0.50724687 0.51615227 0.52482943 0.53323767
##  [31] 0.54144519 0.54951747 0.55730901 0.56497488 0.57247100 0.57986903
##  [37] 0.58711569 0.59433696 0.60136677 0.60822198 0.61482691 0.62134922
##  [43] 0.62759464 0.63379012 0.63987992 0.64588753 0.65171509 0.65740839
##  [49] 0.66301872 0.66852857 0.67393230 0.67919964 0.68434612 0.68941093
##  [55] 0.69440457 0.69925856 0.70401855 0.70868908 0.71319715 0.71757935
##  [61] 0.72193257 0.72621538 0.73039390 0.73451499 0.73851018 0.74247453
##  [67] 0.74637256 0.75019616 0.75385493 0.75746966 0.76101829 0.76448067
##  [73] 0.76783883 0.77112353 0.77437341 0.77759879 0.78077525 0.78389248
##  [79] 0.78698662 0.79000398 0.79295192 0.79589201 0.79880352 0.80169982
##  [85] 0.80453916 0.80729969 0.81003350 0.81270238 0.81532318 0.81785588
##  [91] 0.82033763 0.82277304 0.82517896 0.82756286 0.82987375 0.83215955
##  [97] 0.83440918 0.83660560 0.83877526 0.84092308 0.84299984 0.84506660
## [103] 0.84709858 0.84908373 0.85102530 0.85295242 0.85486688 0.85677131
## [109] 0.85865651 0.86052009 0.86234689 0.86415290 0.86594868 0.86769961
## [115] 0.86942287 0.87112306 0.87279603 0.87445095 0.87607108 0.87765781
## [121] 0.87924387 0.88079341 0.88229600 0.88378751 0.88523848 0.88668425
## [127] 0.88810684 0.88951925 0.89092392 0.89229465 0.89365079 0.89498878
## [133] 0.89630669 0.89761565 0.89890018 0.90016139 0.90139546 0.90261569
## [139] 0.90383215 0.90501957 0.90619359 0.90735952 0.90849789 0.90962775
## [145] 0.91072568 0.91181083 0.91287631 0.91393220 0.91497072 0.91600202
## [151] 0.91702012 0.91800477 0.91898112 0.91995527 0.92089970 0.92183807
## [157] 0.92277137 0.92368239 0.92459113 0.92547490 0.92635356 0.92722062
## [163] 0.92807557 0.92892437 0.92975954 0.93058412 0.93139901 0.93220566
## [169] 0.93300827 0.93380386 0.93457580 0.93533748 0.93608616 0.93683192
## [175] 0.93756835 0.93830154 0.93901952 0.93972571 0.94041590 0.94109818
## [181] 0.94177706 0.94245002 0.94312202 0.94378663 0.94444107 0.94509501
## [187] 0.94573862 0.94637815 0.94700873 0.94763468 0.94825237 0.94885628
## [193] 0.94945831 0.95005533
```