# DATA 605: Assignment 04 - Eigenshoes

CUNY Spring 2021

Philip Tanofsky

21 February 2021

## Introduction: Eigenshoes

This project attempts to generate "eigenshoes", which means to perform matrix transformations on images of shoes using Principal Component Analysis (PCA) using eigenvalues and eigenvectors to display shoe-like images based on the calculated level of variance from the PCA. The project displays the initial JPEG image and then following the matrix transformations displays the eigen-calculated representations of the same images based accounting for at least 80% variability.

Project based on: https://rpubs.com/R-Minator/eigenshoes

```
# Required libraries
library(jpeg)
library(EBImage)
library(OpenImageR)
```

### Retrieve Input File Names

First step, read in the JPEG image names from local directory. At the moment, the project contains 17 images. If the number of JPEG images is different, then change the value assigned to variable $num\_of\_images$.

**Note:** I did not push the JPEG images to my Github account. In order to successfully run this file, copy the *jpg* directory to the same directory as this file, or change the string literal of the variable $img\_dir$ below as needed.

```
# Prepare for Image Processing

# Set constant for number of images to be processed
num_of_images <- 17

# Directory containing the images
# Initial approach pointed to full path of images
#img_dir <- "/Users/philiptanofsky/Documents/School/CUNY/MSDS/Courses/DATA605/Week04/jpg/"
# Edited to move the 'jpg' directory to the same directory as this file, so it's relative pathing
img_dir <- "jpg/"

# Read in the list of file names ending in ".jpg" from the directory
# These are images provided by the professor
files <- list.files(img_dir, pattern="\\.jpg")[1:num_of_images]

#files: Contains the list of jpg file names
```

Variable *files* contains the list of JPEG image file names.

## Display JPEG Function

The function *plot_jpeg* is a utility function to display a JPEG image for the purposes of this project.

Also, global variables are set for image *height*, *width*, and *scale*, along with the number of colors *colors_cnt*, which is set to three because the image is based on the RGB color definition.

```
# Set Adjustment Parameters (default values)
height <- 1200
width <- 2500
scale <- 20
colors_cnt <- 3 # Constant represents R,G,B colors of the jpg images

plot_jpeg <- function(path, add=FALSE) {
  # Read the file
  jpg <- readJPEG(path, native=T)
  # Get the resolution, [x, y]
  res <- dim(jpg)[2:1]
  # Initialize any empty plot are if add == False
  if (!add) {
    plot(1,
         1,
         xlim=c(1, res[1]),
         ylim=c(1, res[2]),
         asp=1,
         type='n',
         xaxs='i',
         yaxs='i',
         xaxt='n',
         yaxt='n',
         xlab='',
         ylab='',
         bty='n')
    rasterImage(jpg,
                1,
                1,
                res[1],
                res[2])
  }
}
```

## Load Image Data

Based on the list of *files* initially retrieved, read in the actual JPEG images, and scale them based on the scale factors defined above.

```
# Based on the default values above
# Creating an arry with dimensions[17, 60, 125, 3] with all zeroes to start
img_arr <- array(rep(0, length(files) * height/scale * width/scale * colors_cnt),
            dim=c(length(files), height/scale, width/scale, colors_cnt))
```

```
# Loop through the image paths
for (i in 1:num_of_images) {
  temp <- resize(readJPEG(paste0(img_dir, files[i])),
                 height/scale,
                 width/scale)
  # Load the scaled images into the image array declared above
  img_arr[i,,,] <- array(temp, dim=c(1, height/scale, width/scale, colors_cnt))
}
```

Variable *img_arr* now contains the JPEG images with the pixels defined in a four-dimensional array.

## Vectorize

Extract the RGB values from the image array containing the original shoe images. The *shoes* dataframe now contains just the color values.

```
# Create matrix of image count, rows (17), and image array dimensions as columns (382500)
flat <- matrix(0, num_of_images, prod(dim(img_arr)))

# Loop through images
for (i in 1:num_of_images) {
  # From image array, pull vector for color Red for given image
  r <- as.vector(img_arr[i,,,1])
  # From image array, pull vector for color Green for given image
  g <- as.vector(img_arr[i,,,2])
  # From image array, pull vector for color Blue for given image
  b <- as.vector(img_arr[i,,,3])
  # Take transpose of the color vectors to make it fit the row length of flat matrix
  flat[i,] <- t(c(r, g, b))
}
# Take transpose of the flat matrix
shoes <- as.data.frame(t(flat))

#dim(shoes)
#382500     17
# As expected, now >300k rows with 17 columns
```

## Actual Plots

Given the initial JPEG images, plot the images for visual inspection.

```
# Plot the original images of the Shoes
par(mfrow=c(3,3))
# mai: set margin in inches
par(mai=c(.3, .3, .3, .3))
# Loop through image array and call the plot_jpeg function
# The image array contains the initial read in of the jpg files
for (i in 1:num_of_images) {
  plot_jpeg(writeJPEG(img_arr[i,,,]))
}
```









Confirmed . . . yes, images of shoes do appear.

## Get Eigen components from correlation structure

Start by scaling the *shoes* dataframe to define the center and scale of the image.

```
scaled <- scale(shoes, center=TRUE, scale=TRUE)
#scaled object contains rows:382500     cols:17
# Clever way to pull the mean of each column using the scale function
#mean.shoe <- attr(scaled, "scaled:center") # saving for classification
# Clever way to pull the standard deviation of each column using the scale function
#std.shoe <- attr(scaled, "scaled:scale") # saving for classification
```

## Calculate Covariance (Correlation)

Based on the *scaled* dataframe, apply the correlation matrix.

```
# Create correlation matrix based on the number of images, in this case 17x17
sigma_ <- cor(scaled)

#sigma_
```

## Get the eigen components

From the correlation matrix, calculate the eigvenvalues and eigenvectors.

```
# Compute the eigenvalues and eigenvectors based on the correlation matrix
myeigen <- eigen(sigma_)
cumsum(myeigen$values) / sum(myeigen$values)
```

```
##  [1] 0.6928202 0.7940449 0.8451073 0.8723847 0.8913841 0.9076338 0.9216282
##  [8] 0.9336889 0.9433872 0.9524455 0.9609037 0.9688907 0.9765235 0.9832209
## [15] 0.9894033 0.9953587 1.0000000
```

## Eigenshoes

To achieve the "eigenshoes" visual, matrix transformations are required on the *scaled* dataframe, containing the scaled color values.

```
# 3 components are selected ... see Conclusion for reasoning
pca_comps <- 3

# Create a matrix based on the first n eigen values
scaling <- diag(myeigen$values[1:pca_comps]^(-1/2)) / (sqrt(nrow(scaled)-1))
# Output matrix to confirm
scaling
```

```
##              [,1]        [,2]        [,3]
## [1,] 0.0004711401 0.000000000 0.000000000
## [2,] 0.0000000000 0.001232586 0.000000000
## [3,] 0.0000000000 0.000000000 0.001735441
```

```
# Output eigenvectors based on the first n eigen values
myeigen$vectors[,1:pca_comps]
```

```
##             [,1]        [,2]        [,3]
##  [1,] -0.2515577 -0.05962807 -0.14114605
##  [2,] -0.2564669  0.22970932 -0.09482706
##  [3,] -0.1974907 -0.34526438 -0.24576573
##  [4,] -0.2391458  0.30516320 -0.13606194
##  [5,] -0.2525203  0.23895414 -0.06096558
##  [6,] -0.2096918 -0.34776361 -0.42324640
##  [7,] -0.2220439 -0.32176935 -0.36923615
##  [8,] -0.2597468  0.13861061 -0.27362524
##  [9,] -0.2242754  0.39008169 -0.17677165
## [10,] -0.2523894  0.26939880  0.02645111
## [11,] -0.2504276  0.23813195  0.14578328
## [12,] -0.2541524 -0.16064493  0.16973475
## [13,] -0.2374627 -0.25443032  0.18739393
## [14,] -0.2431988 -0.17131145  0.34992145
## [15,] -0.2531910 -0.06188346  0.32463869
## [16,] -0.2571186 -0.11980858  0.24383184
## [17,] -0.2513643 -0.10507094  0.30609685
```

```
# Apply the transformation on the scaled matrix which contains the color values.
eigenshoes <- scaled %*% myeigen$vectors[,1:pca_comps] %*% scaling
#dim(eigenshoes)
# 382500      3

# Display the combined visual of the matrix transformation above
for (i in 1:3) {
  imageShow(array(eigenshoes[,i], c(height/scale, width/scale, colors_cnt)))
}
```

```
#imageShow(array(eigenshoes[,1], c(height/scale, width/scale, colors_cnt)))
```

The above image displays the combined visual of the matrix transformations above. The *eigenshoes* array contains three images, based on the three colors of RGB.

## Generate Principal Components

Calculate the principal components based on the initial *img_arr*. The number of principal components will be equal to the number of images under consideration.

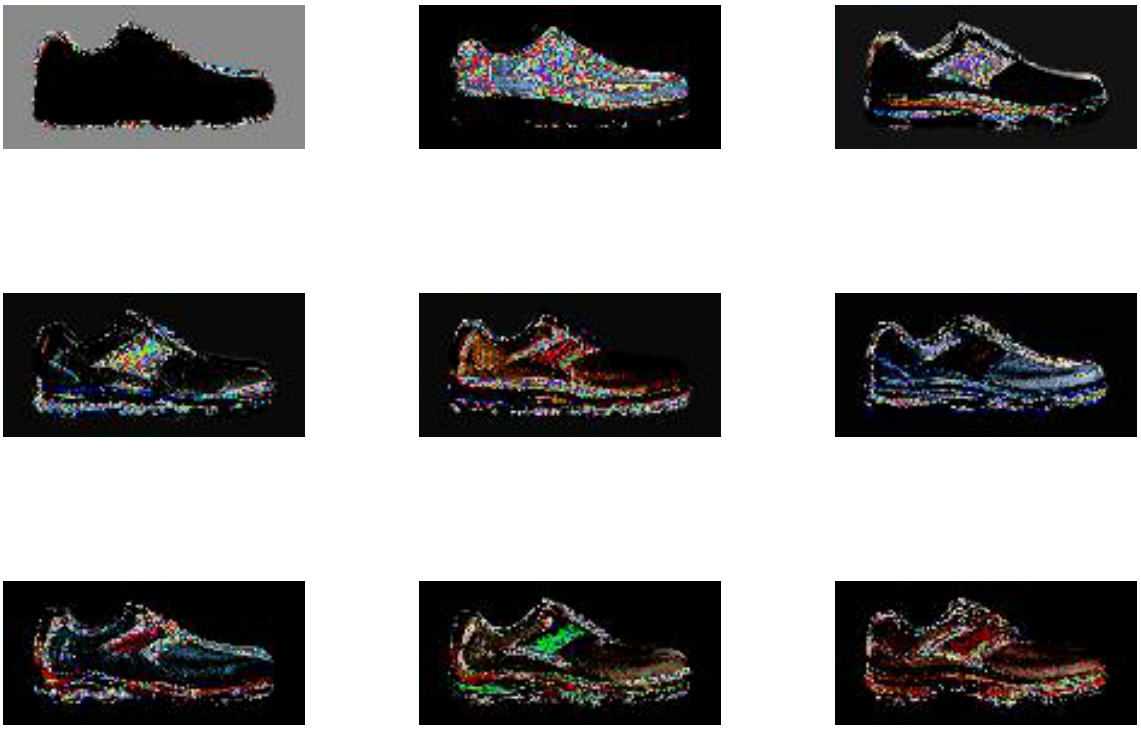
```
# Generate variables
height <- 1200
width <- 2500
scale <- 20
```

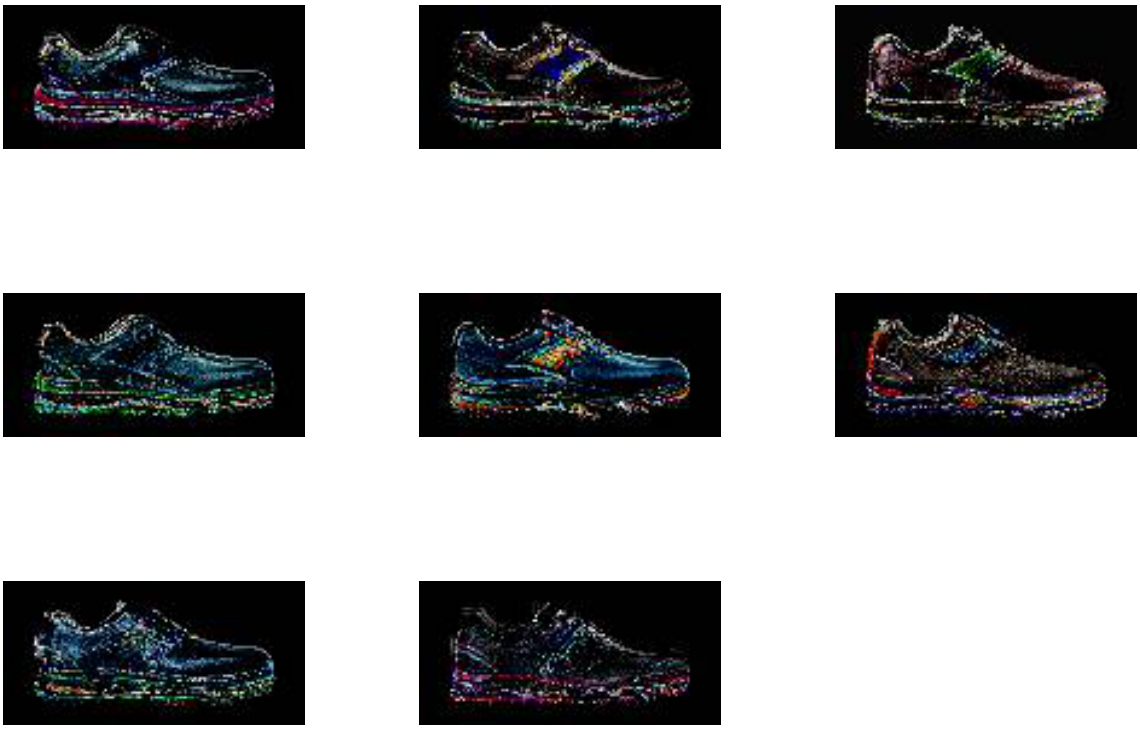
```r
# Start with the image array of the original images
newdata <- img_arr

# Convert the dimensions of the array to n x m, instead of n x m x o x p
dim(newdata) <- c(length(files), height*width*colors_cnt/scale^2)


# Transpose the n x m array, and then calculate the principal components
# By transposing the array, now there are only 17 columns, and thus will produce 17 principal component
mypca <- princomp(t(as.matrix(newdata)), scores=TRUE, cor=TRUE)
# contains 17 components

#mypca

#dim(mypca$scores)
#22500    17
```

## Eigenshoes

To finally generate the display of the "eigenshoes", the transpose of the PCA scores is required to get the matrix back in the form of row count equal to the number of initial images. Based on the transposed PCA scores, the images of "eigenshoes" are rendered as JPEGs.

```r
# Transpose the Principal Component Analysis scores back to original matrix dimensions
mypca2 <- t(mypca$scores)
# Reset the dimensions of the the PCA object to fit the initial array dimensions
dim(mypca2) <- c(length(files), height/scale, width/scale, colors_cnt)
#par(mfrow=c(5,5))
#par(mai=c(.001, .001, .001, .001))
par(mfrow=c(3,3))
par(mai=c(.3, .3, .3, .3))
# Plot the eigenshoes only based on the PCA scores
for (i in 1:num_of_images) {
  plot_jpeg(writeJPEG(mypca2[i,,,], bg="white")) # Complete without reduction
}
```





### Variance Capture

Calculate the level of variability by principal component to determine the number of principal components required to achieve the goal of 80% variability.

```
num_comps <- 17
a <- round(mypca$sdev[1:num_comps]^2 / sum(mypca$sdev^2), 3)
cumsum(a)
```

```
##  Comp.1  Comp.2  Comp.3  Comp.4  Comp.5  Comp.6  Comp.7  Comp.8  Comp.9 Comp.10
##   0.693   0.794   0.845   0.872   0.891   0.907   0.921   0.933   0.943   0.952
## Comp.11 Comp.12 Comp.13 Comp.14 Comp.15 Comp.16 Comp.17
##   0.960   0.968   0.976   0.983   0.989   0.995   1.000
```

## Conclusion

To account for 80% of the variability, the first 3 components are required to exceed the 80% threshold. As displayed above, the first component accounts for 69.3% and the first two combined accounts for 79.4% of the variability. Thus to reach at least 80% variability, the initial three components are required (84.5% variability accounted for) for the "eigenshoes".