Eigneshoes

Shaya Engelman

2024-02-16

Loading Required Libraries

First, we load the necessary libraries for image processing and handling JPEG files.

```r
library(jpeg)
library(EBImage)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.2
```

Listing Image Files

We retrieve a list of sneaker image files from the specified directory. This list will be used for processing a random selection of sneaker images. I stored the base directory as a variable to make it easier to run from other sources by just updating one line.

```r
base_dir <- 'C:/Users/shaya/Downloads/jpg'
files <- list.files(path=base_dir, pattern="\\.jpg")
```

Setting Image Dimensions and Scaling

Here, we define the target dimensions for the images (height and width) and a scaling factor. These parameters will be used to resize the images for further analysis.

```r
height <- 1200
width <- 2500
scale <- 20
```

Function to Plot JPEG Images

A function plot_jpeg is defined to read and plot JPEG images. It takes a file path as input and optionally allows adding the image to an existing plot.

```r
plot_jpeg <- function(path, add = FALSE) {
  jpg <- readJPEG(path, native = TRUE)
  res <- dim(jpg)[2:1]

  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])
}
```

Reading and Resizing Images

In this section, we read each image file, resize it to the specified dimensions, and store the processed images in an array for further analysis. The images are resized to reduce the amount of data that needs to be processed and to ensure that all images have the same dimensions before analysis.

```
im <- array(rep(0, length(files) * height / scale * width / scale * 3),
            dim = c(length(files), height / scale, width / scale, 3))

for (i in seq_along(files)) {
  tmp <- file.path(base_dir, files[i])
  temp <- EBImage::resize(readJPEG(tmp), height / scale, width / scale)
  im[i, , , ] <- array(temp, dim = c(1, height / scale, width / scale, 3))
}
```

Displaying Resized Images

Finally, we plot the resized images in a 3x3 format to visually inspect them.

```
par(mfrow = c(3, 3))
par(mai = c(.3, .3, .3, .3))

for (i in 1:17) {
  plot_jpeg(writeJPEG(im[i, , , ]))
}
```

Image Data Preprocessing and Principal Component Analysis (PCA)

Now, we move on to the analysis of the resized images using Principal Component Analysis (PCA) for dimensionality reduction. PCA is a technique used to emphasize variation and bring out strong patterns in a dataset. It's often used to make data easy to explore and visualize.

In this block, we reshape the image data to fit the requirements of PCA and perform PCA on the transposed matrix of the reshaped data. The result, stored in pca_results, includes the standard deviation of each principal component, and we calculate the proportion of variance explained by these components.

```
newdata <- im
dim(newdata) <- c(length(files), height * width * 3 / scale^2)
pca_results <- princomp(t(as.matrix(newdata)), scores = TRUE, cor = TRUE)
```

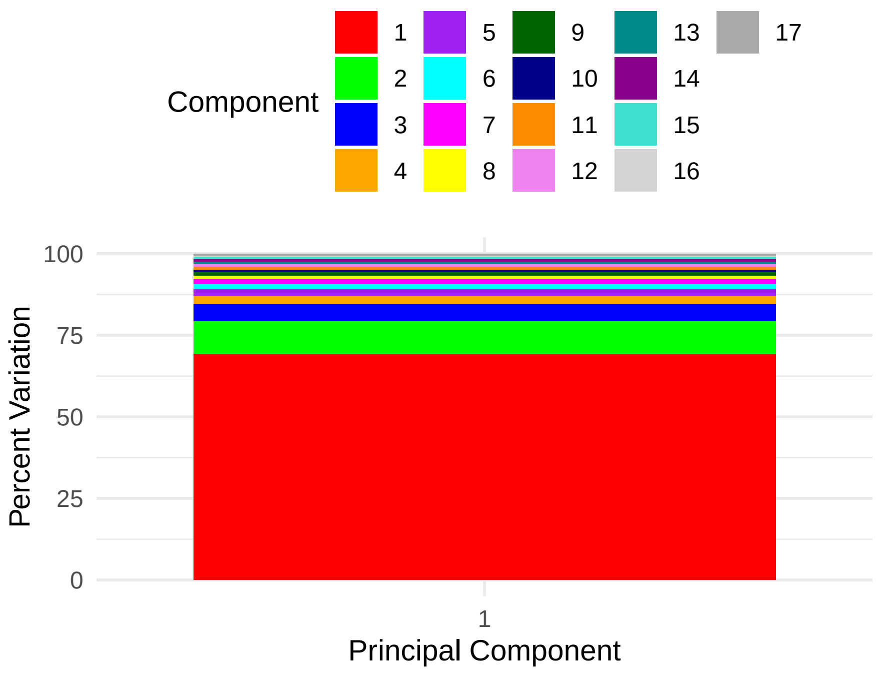
This code calculates the proportion of total variance captured by all the principal components.

```
sum(pca_results$sdev^2 / sum(pca_results$sdev^2))
```

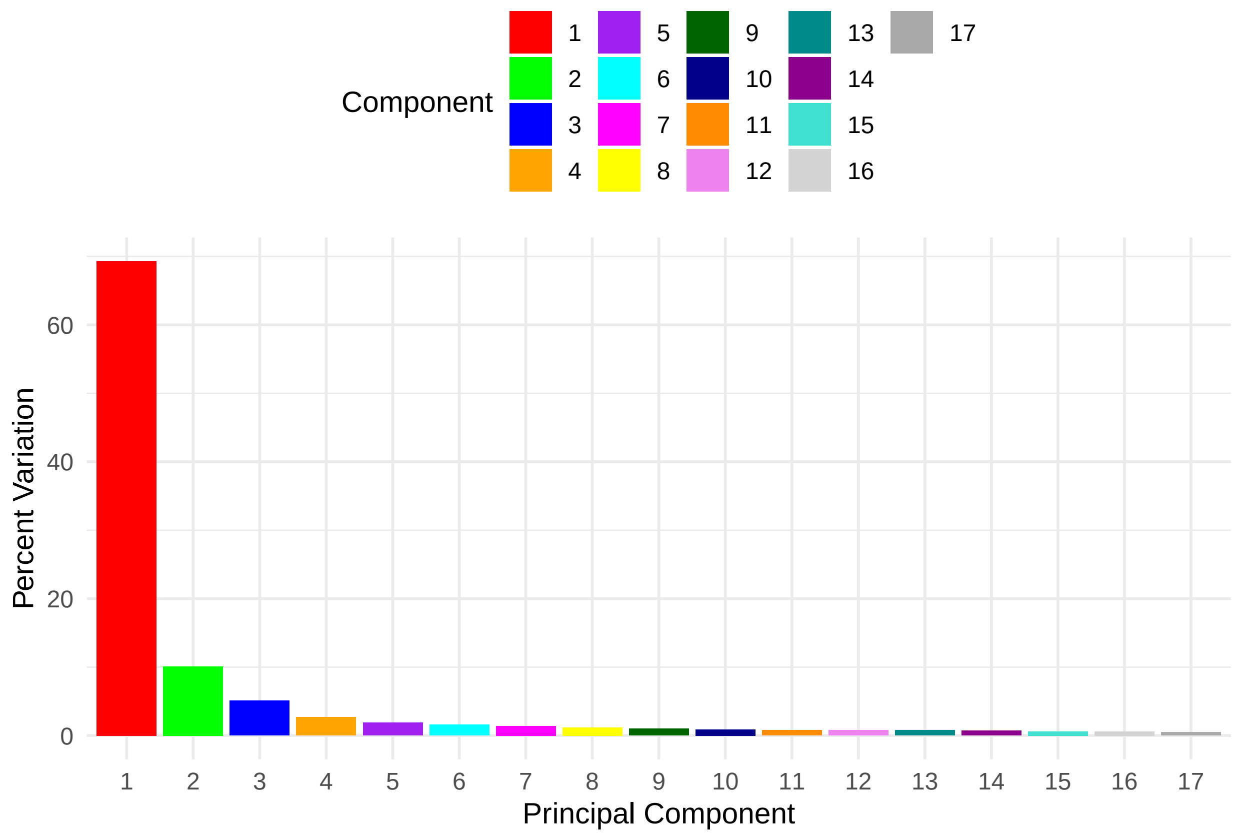
```
## [1] 1
```

In this section, we explore the contribution of individual principal components to the total variance. The plots illustrate the amount of variance captured by each of the eigenshoes represented by each of the principal components. The calculations show the cumulative proportion of variance explained by the first two components and the first six components, respectively.

```r
percentage_variation <- round((pca_results$sdev^2) / sum(pca_results$sdev^2) * 100, 1)

# Create a data frame for ggplot
data <- data.frame(Component = seq_along(percentage_variation),
                   Percentage = percentage_variation)


# Define colors for each component
component_colors <- c("red", "green", "blue", "orange", "purple", "cyan", "magenta", "yellow", "darkgre
                      "darkblue", "darkorange", "violet", "darkcyan", "darkmagenta", "turquoise", "ligh


ggplot(data, aes(x = factor(1), y = Percentage, fill = factor(Component))) +
  geom_bar(stat = "identity", position = position_stack(reverse = TRUE)) +
  scale_fill_manual(values = component_colors) +
  labs(x = "Principal Component", y = "Percent Variation",
       fill = "Component") +
  theme_minimal() +
  theme(legend.position = "top",
        plot.margin = margin(0.5, 1, 0.5, 1, "in"))
```



```r
ggplot(data, aes(x = factor(Component), y = Percentage, fill = factor(Component))) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = component_colors) +
  labs(x = "Principal Component", y = "Percent Variation",
      fill = "Component") +
  theme_minimal() +
  theme(legend.position = "top")
```



```r
pca_components <- pca_results$sdev^2 / sum(pca_results$sdev^2)
sum(pca_components[1:2])
```

```
## [1] 0.7940449
```

```r
sum(pca_components[1:6])
```

```
## [1] 0.9076338
```

Here, we reshape the PCA scores to the original image dimensions for visualization purposes. And then, finally, we display a grid of the images representing the principal components. These images showcase the patterns captured by the most significant components in the PCA analysis.

```r
pca_scores_reshaped <- t(pca_results$scores)
dim(pca_scores_reshaped) <- c(length(files), height / scale, width / scale, 3)

par(mfrow = c(5, 5))
par(mai = c(.001, .001, .001, .001))
```

```r
for (i in 1:17) {
  plot_jpeg(writeJPEG(pca_scores_reshaped[i, , , ], quality = 1, bg = "white"))
}
```

To ensure that our results aren't being influenced too much by the resolution we picked, we can rerun the analysis on a slightly higher and slightly lower resolution to see if anything specific stands out.

```r
# Adjust the resolution
higher_height <- 1800
higher_width <- 3700

# Rerun PCA with higher resolution
higher_im <- array(rep(0, length(files) * higher_height / scale * higher_width / scale * 3),
                   dim = c(length(files), higher_height / scale, higher_width / scale, 3))

for (i in seq_along(files)) {
  tmp <- file.path(base_dir, files[i])
  temp <- EBImage::resize(readJPEG(tmp), higher_height / scale, higher_width / scale)
  higher_im[i, , , ] <- array(temp, dim = c(1, higher_height / scale, higher_width / scale, 3))
}

# Reshape and run PCA
higher_newdata <- higher_im
dim(higher_newdata) <- c(length(files), higher_height * higher_width * 3 / scale^2)
higher_pca_results <- princomp(t(as.matrix(higher_newdata)), scores = TRUE, cor = TRUE)
```

```r
# Calculate the number of components needed for 80% variance
higher_components <- (higher_pca_results$sdev^2 / sum(higher_pca_results$sdev^2))
```

```r
# Adjust the resolution
lower_height <- 800
lower_width <- 1700

# Rerun PCA with lower resolution
lower_im <- array(rep(0, length(files) * lower_height / scale * lower_width / scale * 3),
                  dim = c(length(files), lower_height / scale, lower_width / scale, 3))

for (i in seq_along(files)) {
  tmp <- file.path(base_dir, files[i])
  temp <- EBImage::resize(readJPEG(tmp), lower_height / scale, lower_width / scale)
  lower_im[i, , , ] <- array(temp, dim = c(1, lower_height / scale, lower_width / scale, 3))
}

# Reshape and run PCA
lower_newdata <- lower_im
dim(lower_newdata) <- c(length(files), lower_height * lower_width * 3 / scale^2)
lower_pca_results <- princomp(t(as.matrix(lower_newdata)), scores = TRUE, cor = TRUE)

# Calculate the number of components needed for 80% variance
lower_components <- (lower_pca_results$sdev^2 / sum(lower_pca_results$sdev^2))
```

When printing the variance captured by the first 2 and the first 6 of the eigenimages of all three of the analyses respectively, we can see that there are only very slight differences. This means we do not have to worry about the resolution we chose influencing the results. (Interestingly, the resolution we chose seems to be a slightly better model than both the higher and lower resolution options, something I find extremely strange).

```r
#original results
sum(pca_components[1:2])
```

```
## [1] 0.7940449
```

```r
sum(pca_components[1:6])
```

```
## [1] 0.9076338
```

```r
#higher resolution
sum(higher_components[1:2])
```

```
## [1] 0.7890335
```

```r
sum(higher_components[1:6])
```

```
## [1] 0.9050178
```

```r
#lower resolution
sum(lower_components[1:2])
```

```
## [1] 0.7941642
```

```r
sum(lower_components[1:6])
```

```
## [1] 0.906253
```