Image Compression using Dimension Reduction aka PCA

Divij Pherwani

Table of Contents

Image compression with principal component analysis is an application of the dimension reduction technique. The image can be expressed as a matrix of pixel represented by the RGB value. Reduction in Dimensions will reduce the image size while trying to maintain the quality. The image used in this analysis is of Warsaw skyscrapers in center. The picture is clicked close to central railway station at night. The image is free for commercial use and there is no attribution required. The source of image is https://pixabay.com/photos/warsaw-central-railway-station-3461016/.

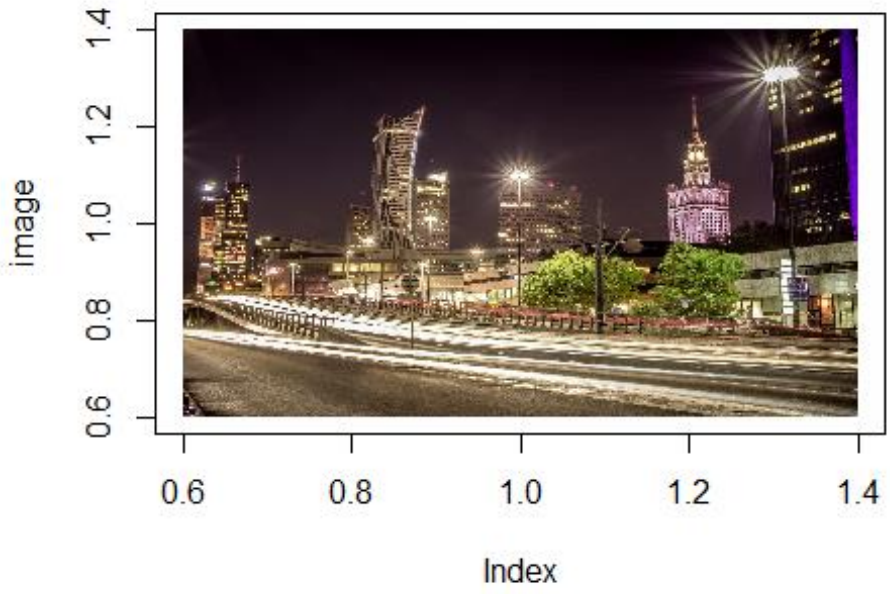
Initialization

The project uses libraries such as jpeg, factoextra, magick, ggplot2 and imager. The working directory is set to source file location as the image is located in the same directory as the R file.

```
## Loading libraries and setting working directory as source file location
library(jpeg)
library(factoextra)
library(magick)
library(imager)
library(data.table)
library(ggplot2)
setwd(dirname(rstudioapi::getSourceEditorContext()$path))
```

The image is read using the readJPEG function and stored in the photo variable. The image is later plotted using plot and roasterImage function.

```
# Reading the image and loading the image in a variable
photo <- readJPEG("Warsaw.jpg")
plot(1,type="n", ylab = "image")
rasterImage(photo, 0.6,0.6,1.4,1.4)
```



Checking class of the photo variable and it is an array.

```
class(photo)

## [1] "array"
```

Summary analysis of photo

```
summary(photo)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1137  0.2196  0.3008  0.4275  1.0000
```

Further analysis of photo array using str function

```
str(photo)

##  num [1:1233, 1:1920, 1:3] 0.102 0.102 0.098 0.098 0.098 ...
```

Analysing the dimensions of the 3D array of the photo

```
dim <- dim(photo)
dim

## [1] 1233 1920    3
```

Segregating the values of Red, Green and Blue from the photo variable and storing it in R, G and B variable. PCA will be performed on each of the extracted individual color matrix value from the RGB color value scheme.

```
# Extracting values to form RGB matrix
R <- photo[,,1]
G <- photo[,,2]
B <- photo[,,3]
```

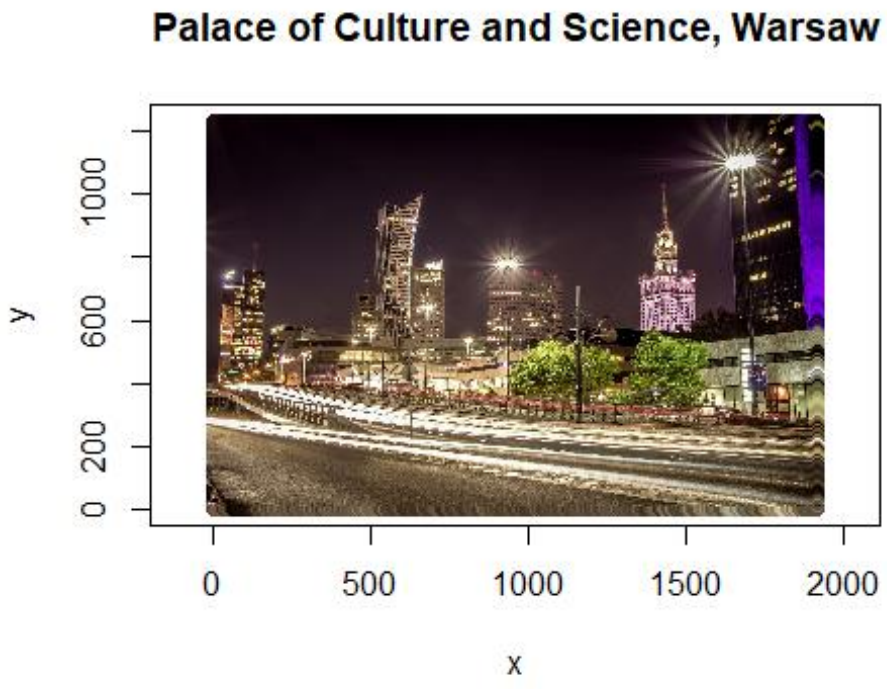
Making a table with values of x,y, red, green and blue. The table values will be later used to reconstruct image from data.

```
# Making RGB table and displaying it
rgb_table<-data.frame(x=rep(1:dim[2], each=dim[1]),  y=rep(dim[1]:1, dim[2]),
r.value=as.vector(R),  g.value=as.vector(G), b.value=as.vector(B))
head(rgb_table)
```

```
##   x    y    r.value    g.value    b.value
## 1 1 1233 0.10196078 0.05490196 0.07058824
## 2 1 1232 0.10196078 0.05490196 0.07058824
## 3 1 1231 0.09803922 0.05098039 0.06666667
## 4 1 1230 0.09803922 0.05098039 0.06666667
## 5 1 1229 0.09803922 0.05098039 0.06666667
## 6 1 1228 0.09803922 0.05098039 0.06666667
```

Using the above created table to plot the photo from the data. The image reconstruction using the data is an interesting phenomenon.

```
plot(y~x, data=rgb_table, main="Palace of Culture and Science, Warsaw",
col=rgb(rgb_table[c("r.value", "g.value", "b.value")]), asp=1)
```



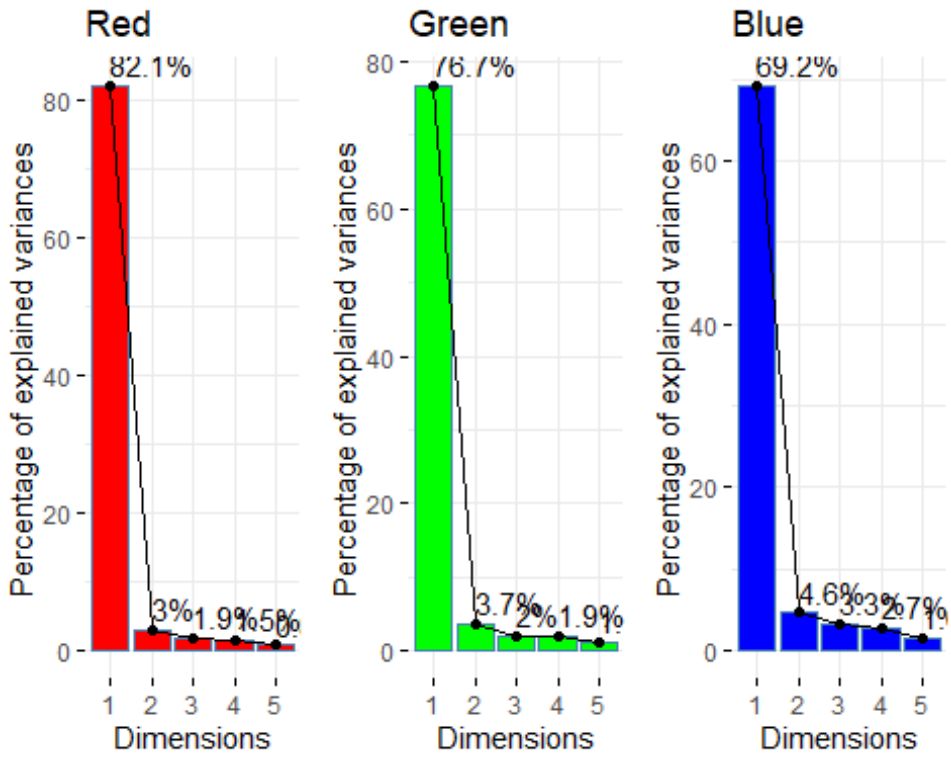
Applying PCA on each of the extracted color matrix values.

```
# PCA for each color component
pca_R <- prcomp(R, center = FALSE)
pca_G <- prcomp(G, center = FALSE)
pca_B <- prcomp(B, center = FALSE)
```

Scree Plots and Cummilative Variation Plots

In percentage of variance explained, the 1 principal component for Red explains ~ 82 % of total variance, for Green ~ 77% of total variance and for Blue ~ 69% of total variance.

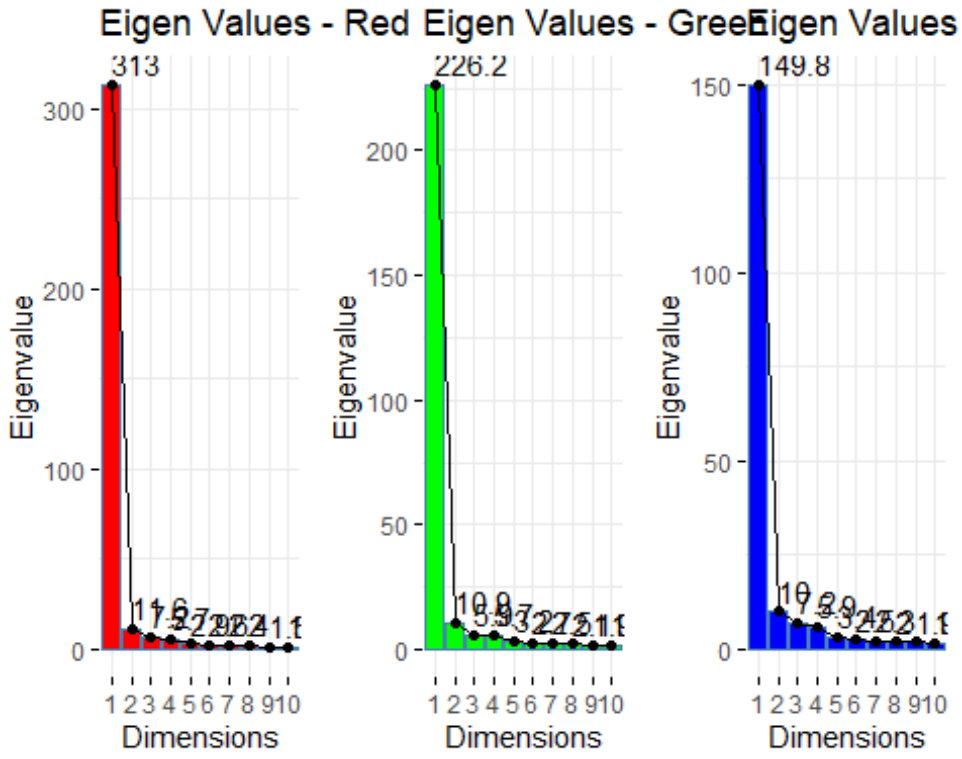
```
# Percentage of Explained variance for Red, Green and Blue
r1 <- fviz_eig(pca_R, addlabels = TRUE, barfill = "RED", ncp = 5, main =
"Red")
g1 <- fviz_eig(pca_G, addlabels = TRUE, barfill = "GREEN", ncp = 5, main =
"Green")
b1 <- fviz_eig(pca_B, addlabels = TRUE, barfill = "BLUE", ncp = 5, main =
"Blue")
gridExtra::grid.arrange(r1,g1,b1, nrow = 1, ncol = 3)
```



We know that we need to keep eigen values greater than 1 for factor analysis. From the RGB perspective, the eigen value for first component is significantly larger. The eigen values for first component for Red is 313, for Green is ~ 226 and for Blue is ~ 150.

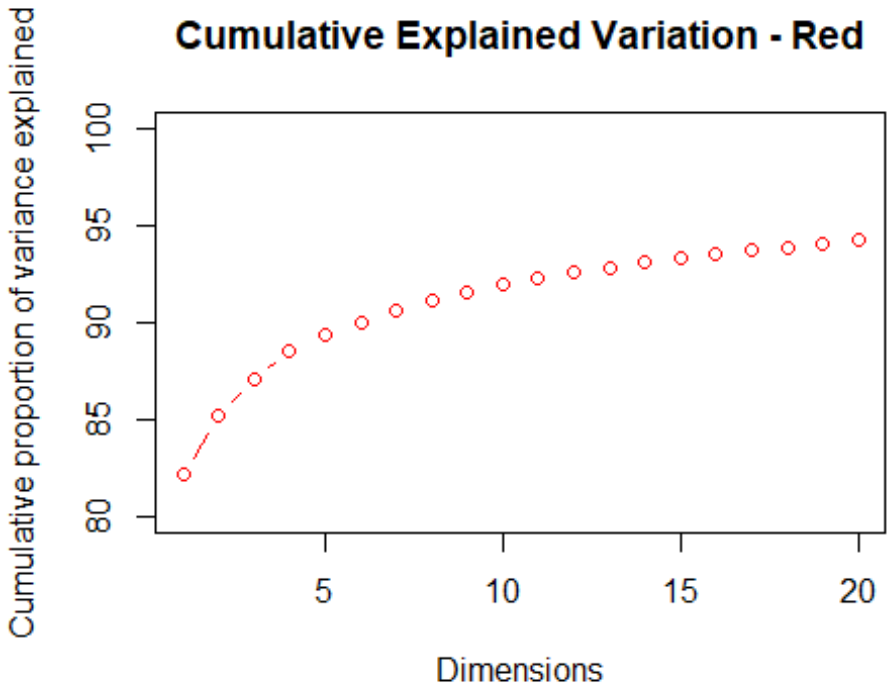
```
# Eigen Values vs Dimensions for Red, Green and Blue
r2 <- fviz_eig(pca_R, addlabels = TRUE, choice = "eigenvalue", barfill =
"RED", ncp = 10, main = "Eigen Values - Red")
g2 <- fviz_eig(pca_G, addlabels = TRUE, choice = "eigenvalue", barfill =
```

```
"GREEN", ncp = 10, main = "Eigen Values - Green")
b2 <- fviz_eig(pca_B, addlabels = TRUE, choice = "eigenvalue", barfill =
"BLUE", ncp = 10, main = "Eigen Values - Blue")
gridExtra::grid.arrange(r2,g2,b2, nrow = 1, ncol = 3)
```

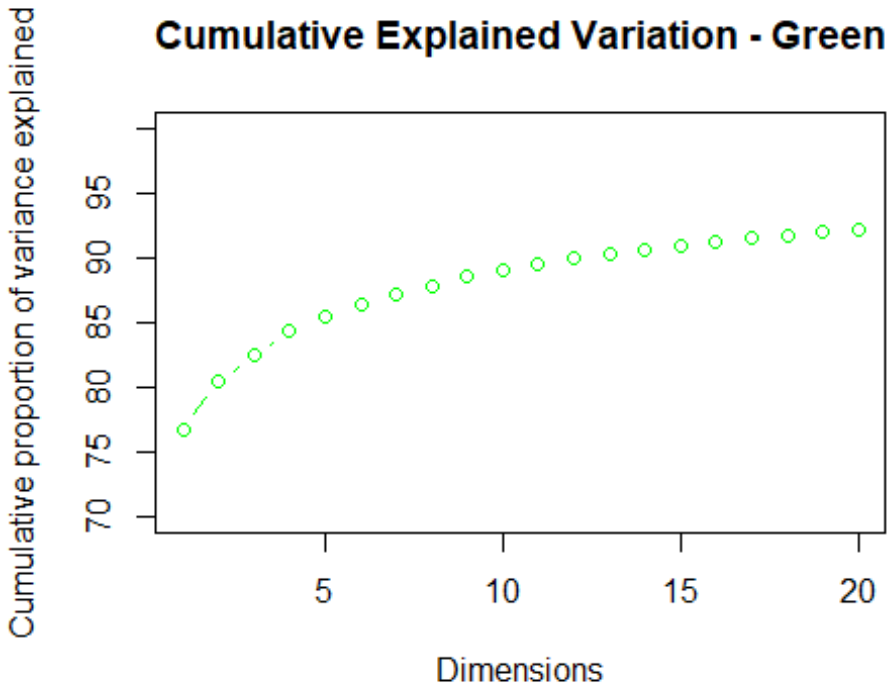


The cumulative explained variation for Red is the highest followed by Blue and then by Green.

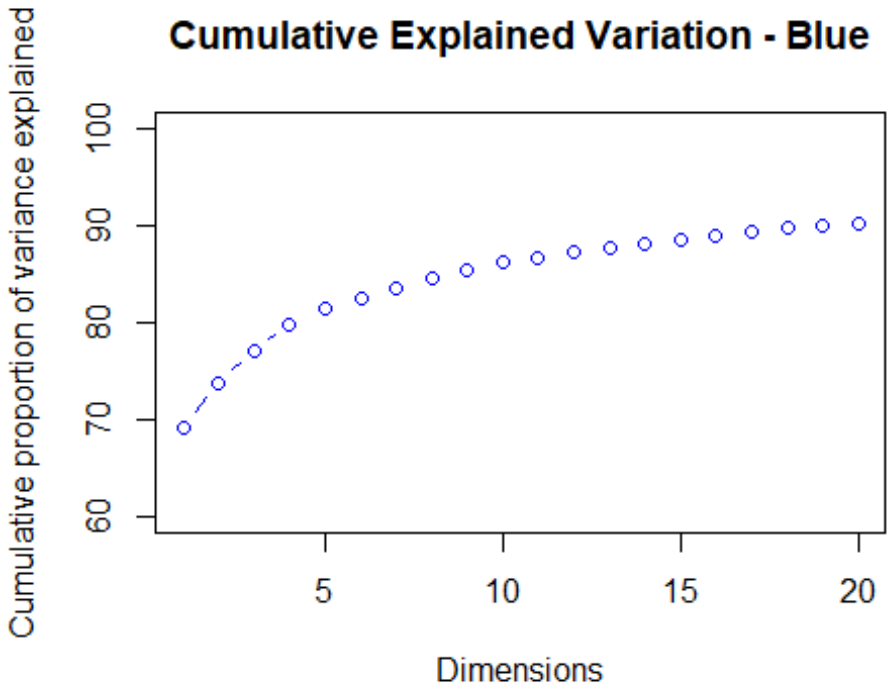
```
# Cumulative Proportion of variance Explained for Red, Green and Blue
r3 <- plot(100 * cumsum(pca_R$sdev^2 / sum(pca_R$sdev^2))[1:20],ylim =
c(80,100), xlab = "Dimensions", ylab = "Cumulative proportion of variance
explained", type="b", main = "Cumulative Explained Variation - Red", col =
"RED")
```




```
g3 <- plot(100 * cumsum(pca_G$sdev^2 / sum(pca_G$sdev^2))[1:20],ylim =
c(70,100), xlab = "Dimensions", ylab = "Cumulative proportion of variance
explained", type="b", main = "Cumulative Explained Variation - Green", col =
"Green")
```




```
b3 <- plot(100 * cumsum(pca_B$sdev^2 / sum(pca_B$sdev^2))[1:20],ylim =
c(60,100), xlab = "Dimensions", ylab = "Cumulative proportion of variance
explained", type="b", main = "Cumulative Explained Variation - Blue", col =
"Blue")
```

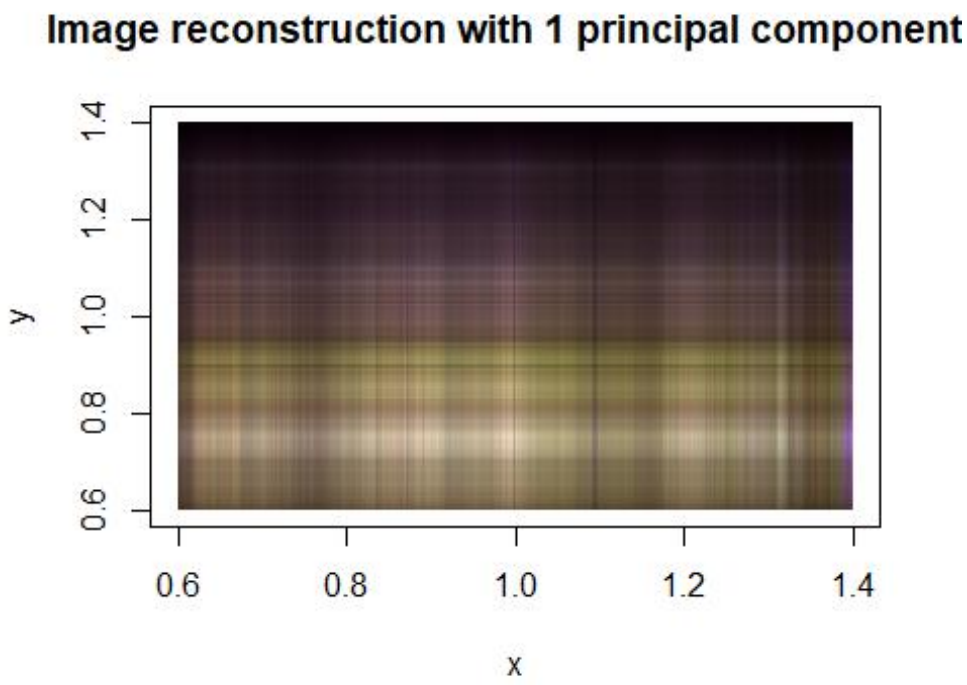


Combing the PCA values of Red, Green and Blue in a single list

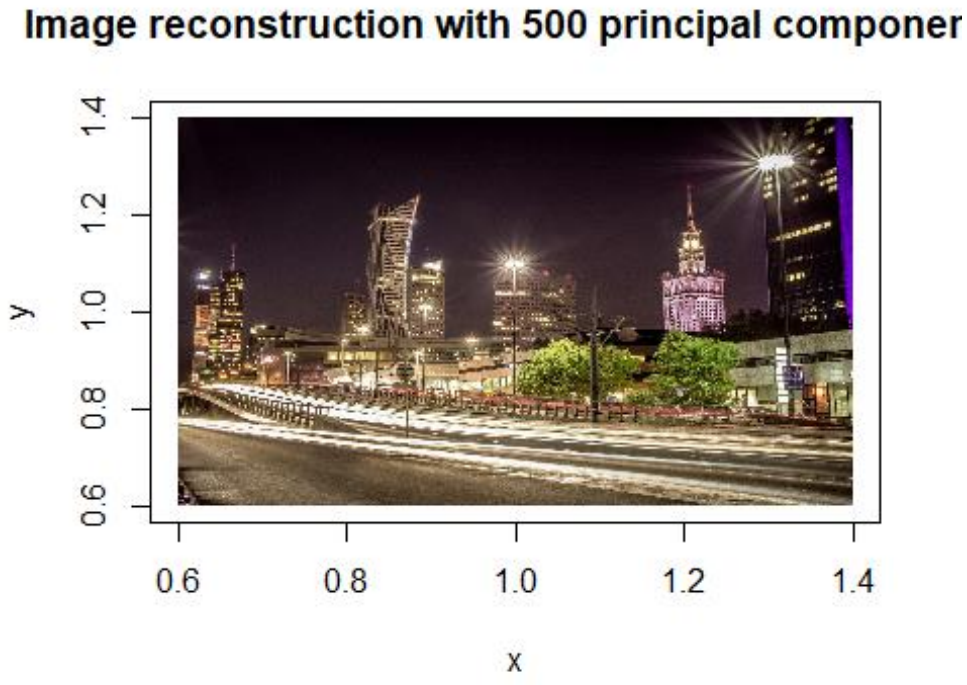
```
# Making a single list of PCA values for RGB
pca_rgb <- list(pca_R, pca_G, pca_B)
```

The compress function below takes a parameter n which is the number of required dimensions as output. The function will reduce the dimensions and generate the required output. There is an option to write the image which will save the image in local device.

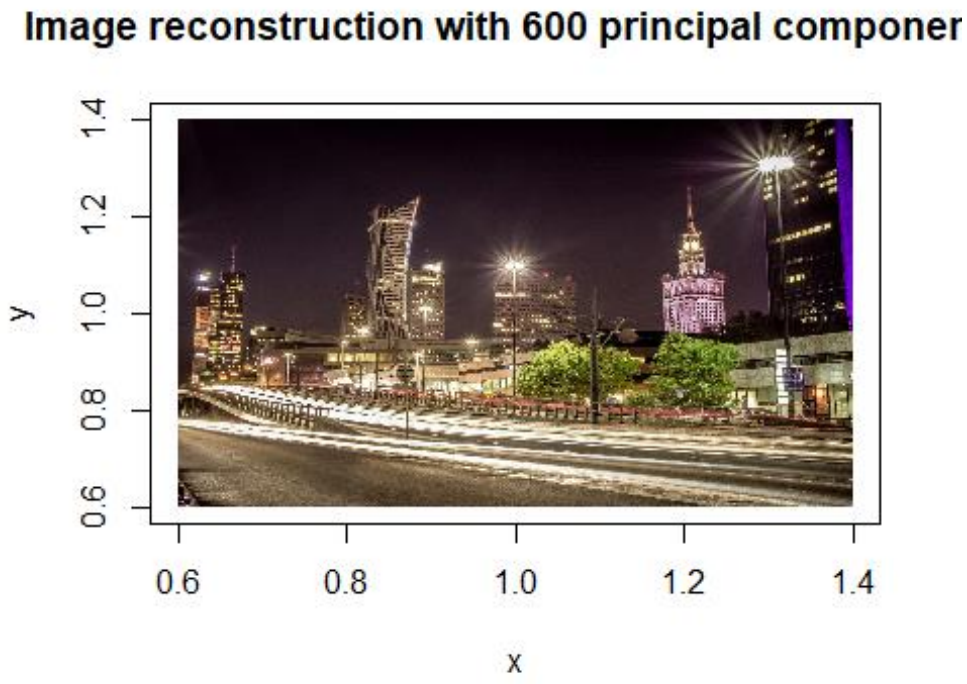
```
# Making a compress function to reduce dimensions of image and reduce size as
a result
compress <- function(n, write = FALSE)

{
  pca_img <- sapply(pca_rgb, function(k){compressed <- k$x[, 1:n] %*%
t(k$rotation[, 1:n])}, simplify='array')
  plot(1, type='n', main = paste("Image reconstruction with", n, "principal
components"), xlab = "x", ylab = "y")
  rasterImage(image_read(pca_img), 0.6, 0.6, 1.4, 1.4)
  if (write == TRUE)
  {
    writeJPEG(pca_img, paste("Image reconstruction with", round(n, 0),
"principal components.jpg"))
  }
}

compress(1)
```




```
compress(500)
```



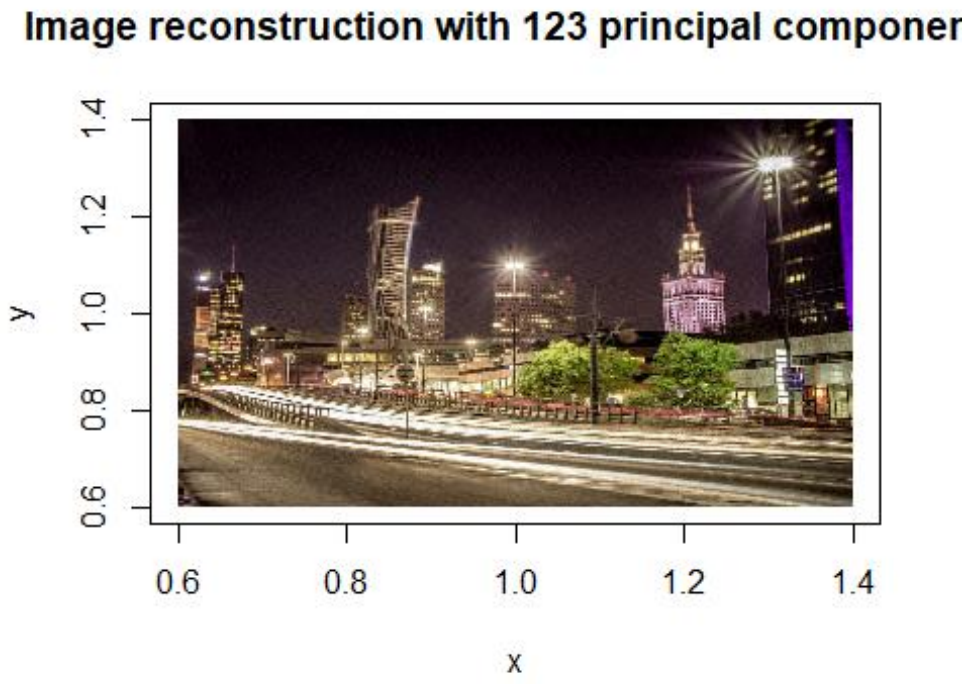

```
compress(600)
```




```
# Running compress function few times to check the quality of the images. The
principal components are run starting from 100 till 1233 with an increment at
each step of 180.

#for (i in c(seq(100, nrow(photo), by=180)))
#{
 # compress(i)
#}
```

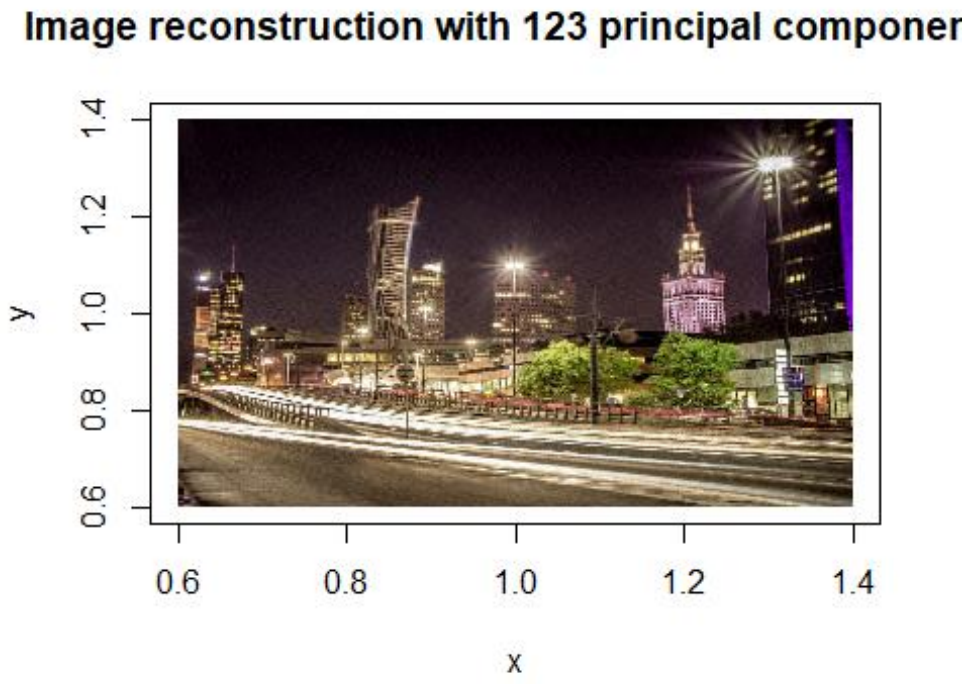
The compressor function is an extension of compress function which takes n as input. The n stands for the n% of the image original dimensions. For example, we want to reduce the image to 10% of original dimensions then n will be 10.

```
compressor <- function(n)
{
  rank <- nrow(photo)
  comp <- n*rank/100
  comp <- round(comp,0)
  print(comp)
  compress(comp)

}

compressor(10)

## [1] 123
```



Testing the compress function to compress the image to 123 prinicial components out of possible 1233.

```
compress(123, write = TRUE)
```



Comparing compress function with resize functionality. I use the resize functionality to reduce image dimensions to 10% of original dimensions. The quality obtained from the resize functionality is way worse than reduction to 10% of total dimensions in the compress function (123 prinicipal components). We can clearly see that the quality is retained in PCA after the image is reformed.

```
# Using Resize functionality
im <- load.image("Warsaw.jpg")
plot(resize(im, round(width(im)/10),round(height(im)/10)))
```

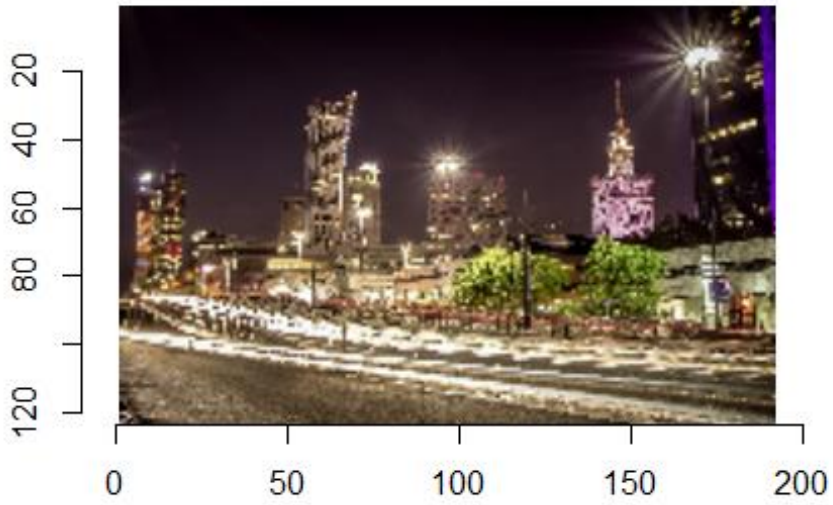


Image Size Analysis

Using size reduction calculations we can see that reducing principal components to 200, 500, 700 and 1000 reduces the size of the image more than 50% while maintaining the quality.

```
# Doing Image Size change analysis using principal components 200, 500, 700
and 1000

components <- c(200, 500, 700, 1000)
size <- c()

original <- file.info("Warsaw.jpg")$size / 1000

for (n in components)
{
  pca_img <- sapply(pca_rgb, function(k){compressed <- k$x[, 1:n] %*%
t(k$rotation[, 1:n])}, simplify='array')
  {
     writeJPEG(pca_img, paste("Temp Last Run.jpg"))
  }
  size <- c(size, file.info("Temp Last Run.jpg")$size / 1000)
}

unlink("Temp Last Run.jpg")
```

```
#The number of principal components and the percentage reduction in size

data.frame("Principal Components" = components, "change in size by percent" =
100* ((size/original)- 1))

##   Principal.Components change.in.size.by.percent
## 1                  200                 -45.20500
## 2                  500                 -41.86780
## 3                  700                 -43.46773
## 4                 1000                 -47.95633
```

Summary

I loaded the image into a variable, extracted the RGB color scheme matrix, performed principal component analysis for dimension reduction and reconstructed the image using the data. There are many real world applications of PCA technique. For the image compression we observed that with the reduction in components, the size of the image was reduced. The quality of image was preserved as much as possible, for example yhe quality of image with 500 principal components and 700 was pretty similar. That is an advantage of using PCA technique over standard re-sizing functionality.

References:

1. Image Compression with Principal Component Analysis and R, https://medium.com/@AaronSchlegel/image-compression-with-principal-component-analysis-f6e8e73065af

2. University of Warsaw, Unsupervised Learning Course by dr Jacek Lewkowicz