```
---
title: "EBC4223 - Assignment 4"
author: "Quang Phong & Robert Agatić"
date: "3/14/2022"
output:
  html_document: default
  pdf_document: default
---
```

# INTRODUCTION

In this assignment, we will replicate some of the image feature extraction techniques discussed in the papers of session 5.

We have been provided with a dataset containing 2,184 color images of clothing items. Labels are also available, identifying the brand and product type of an image. We will use these labels to train models aimed at **identifying different brands and product types based on image characteristics.**

Because the full database might be too large for our PCs to handle, we select a random subset of images (and labels) and perform the assignment on that subset.

# DATA PREPARATION

**Question 1: Load libraries**

Four libraries are required for this analysis: pixmap, imager, wvtool and colordistance. These contain the functions required to extract image features.

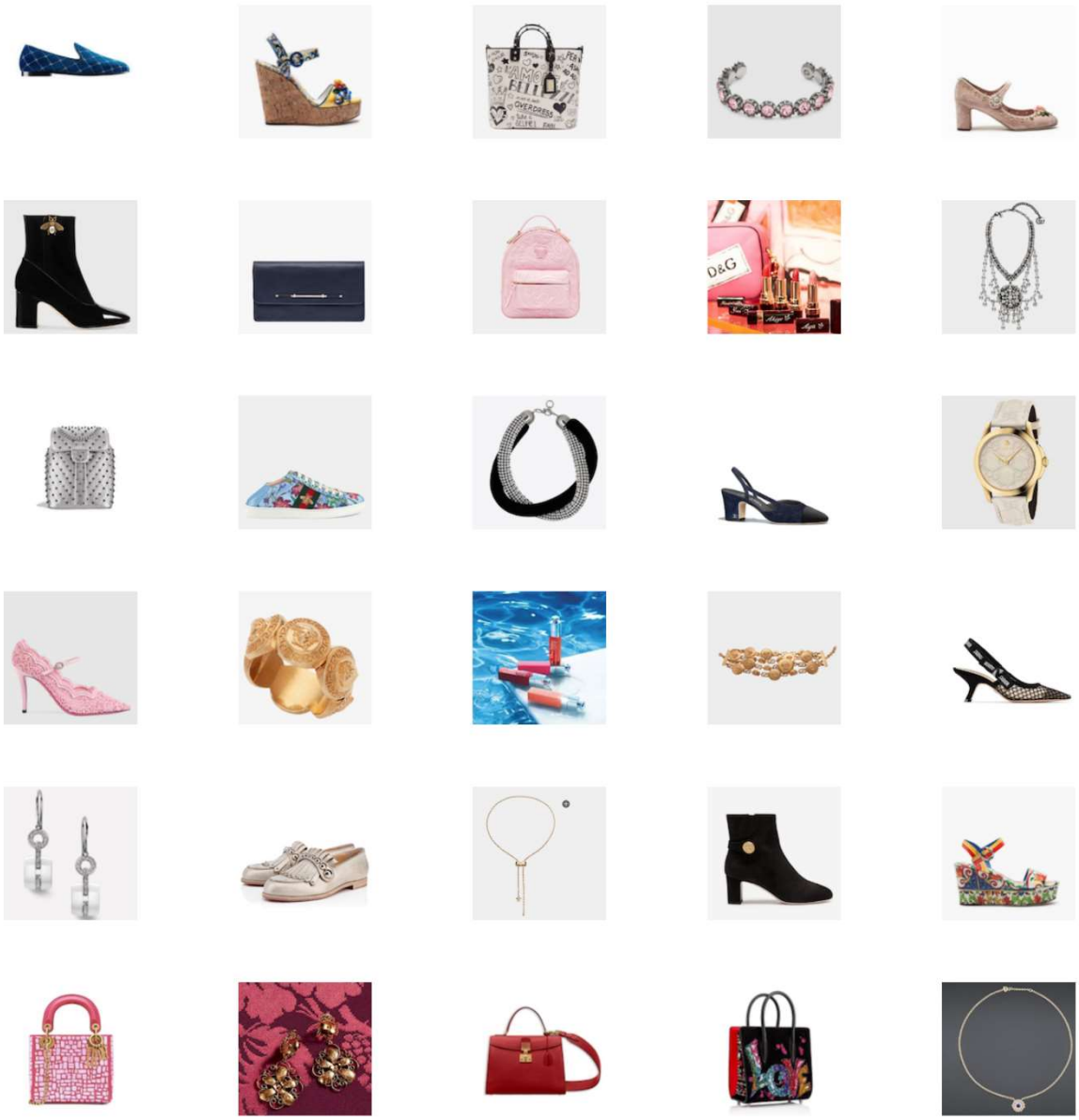
```{r, message=FALSE, warning=FALSE}
# Load necessary libraries
library(pixmap)
library(imager)
library(wvtool)
library(colordistance)
library(dplyr)
library(caret)
library(e1071)
library(LiblineaR)
library(MLmetrics)
library(tinytex)
```

**Question 2: Import images**

Here, we import the images using image(). However, we also apply the rm.alpha() function to remove the alpha channel (capturing transparency) from the image. We only want images containing three channels (i.e. three colors)

```{r}
# Produce a character vector of the names of files in the named directory
allFiles <- list.files(path = "Images", pattern = ".png", full.names = T)

# Choose a random subset of 250 files in the list
sampleNumber <- 250

set.seed(1010)
selectedFiles <- sample(allFiles, sampleNumber)

# Look at the first 10 of 250 random files
selectedFiles[1:10]

# Import 250 images using load.image() in package "imager"
pictures <- lapply(selectedFiles, FUN = function(f) load.image(f))

# Remove alpha channel
pictures2 <- lapply(pictures, FUN = function(f) rm.alpha(f))
```

```
# View the first 30 images
par(mfrow=c(6, 5)) # set the plotting area into a 6*5 array to save space
for (x in pictures2[1:30]) {
  plot(x, axes = FALSE) # remove axes
}
```




```
# Make sure the images in list pictures2 only contain three channels
pictures[[1]]
pictures2[[1]]
```

```
Image. Width: 150 pix Height: 150 pix Depth: 1 Colour channels: 4
Image. Width: 150 pix Height: 150 pix Depth: 1 Colour channels: 3
```

As can be seen, images in pictures2 list only have 3 colour channels.

**Question 3: Import label dataset**

Here we import the label dataset using csv(). We will append this dataframe with image features for each image as additional columns.

```{r}
label <- read.csv("style.csv")

# Look at the first 3 rows of label
label[1:3, ]
```

| | brand_name <chr> | brand_label <int> | product_na... <chr> | product_label <int> |
|---|---|---|---|---|
| 1 | Christian Louboutin | 0 | shoes | 0 |
| 2 | Christian Louboutin | 0 | shoes | 0 |
| 3 | Christian Louboutin | 0 | shoes | 0 |

3 rows | 1-5 of 5 columns

```
# Remove "Images/" string in selectedFiles list so that we can have matching file
names with the label data frame
selectedFiles2 <- gsub(".*/", "", selectedFiles)

# Take the labels for selected files
selectedLabels <- subset(label, file %in% unlist(selectedFiles2))

selectedLabels[1:3, ]
dim(selectedLabels)
```

```
[1] 250   5
```

# DATA ANALYSIS

**Question 4: Extract color histograms**

RGB and HSV color histograms can be extracted using the getImageHist(). One particularity is that it requires images loaded by the loadImage() function. Hence, we will load the images using this function as well. As numbers of bins, we take 3. The arguments hsv (TRUE/FALSE) determines whether an RGB or HSV histogram is produced. The first three columns of the function output contain the histogram for each colour channel. We create a matrix with 81 columns (3*3*3) for each of the RGB and HSV histograms, and add these to the labels dataset.

```{r, warning=FALSE, message=FALSE}
# Load images using loadImage() from package "colorImage"
pictures3 <- lapply(selectedFiles, FUN = function(f) loadImage(f))

# Create an empty matrix first. This matrix will store 162 columns from rgb and hsv
for each picture later.
mat1 <- matrix(, nrow=0, ncol = 162)

for (picture in pictures3) {

  # Generate RGB histogram
  rgb <- getImageHist(picture, bins = 3, hsv = FALSE, plotting = FALSE)

  # Generate HSV histogram
  hsv <- getImageHist(picture, bins = 3, hsv = TRUE, plotting = FALSE)

  # Combine the first 3 columns of the RGB histogram output and the first 3 columns
of HSV histogram output as one list
  list1 <- c(rgb[, 1], rgb[, 2], rgb[, 3],
             hsv[, 1], hsv[, 2], hsv[, 3])

  # Turn that list into the matrix of 81*2 columns
  mat1_ <- matrix(unlist(list1), ncol = 162)

  # Add that newly created matrix to the big matrix
  mat1 <- rbind(mat1, mat1_)
}
```

```{r}
# Add the selected file names to the matrix so that we can merge it with the labels
data set later
selectedFiles3 <- data.frame(cbind(selectedFiles2, mat1))

# Set column names
colnames(selectedFiles3) <- c("name",
                              "rgb1", "rgb2","rgb3","rgb4","rgb5","rgb6","rgb7"
,"rgb8","rgb9","rgb10",
                              "rgb11", "rgb12","rgb13","rgb14","rgb15","rgb16"
,"rgb17","rgb18","rgb19","rgb20",
                              "rgb21", "rgb22","rgb23","rgb24","rgb25","rgb26"
,"rgb27","rgb28","rgb29","rgb30",
                              "rgb31", "rgb32","rgb33","rgb34","rgb35","rgb36"
,"rgb37","rgb38","rgb39","rgb40",
                              "rgb41", "rgb42","rgb43","rgb44","rgb45","rgb46"
,"rgb47","rgb48","rgb49","rgb50",
                              "rgb51", "rgb52","rgb53","rgb54","rgb55","rgb56"
,"rgb57","rgb58","rgb59","rgb60",
                              "rgb61", "rgb62","rgb63","rgb64","rgb65","rgb66"
,"rgb67","rgb68","rgb69","rgb70",
                              "rgb71", "rgb72","rgb73","rgb74","rgb75","rgb76"
,"rgb77","rgb78","rgb79","rgb80",
```

```
                                "rgb81",
                                "hsv1", "hsv2","hsv3","hsv4","hsv5","hsv6","hsv7"
,"hsv8","hsv9","hsv10",
                                "hsv11", "hsv12","hsv13","hsv14","hsv15","hsv16"
,"hsv17","hsv18","hsv19","hsv20",
                                "hsv21", "hsv22","hsv23","hsv24","hsv25","hsv26"
,"hsv27","hsv28","hsv29","hsv30",
                                "hsv31", "hsv32","hsv33","hsv34","hsv35","hsv36"
,"hsv37","hsv38","hsv39","hsv40",
                                "hsv41", "hsv42","hsv43","hsv44","hsv45","hsv46"
,"hsv47","hsv48","hsv49","hsv50",
                                "hsv51", "hsv52","hsv53","hsv54","hsv55","hsv56"
,"hsv57","hsv58","hsv59","hsv60",
                                "hsv61", "hsv62","hsv63","hsv64","hsv65","hsv66"
,"hsv67","hsv68","hsv69","hsv70",
                                "hsv71", "hsv72","hsv73","hsv74","hsv75","hsv76"
,"hsv77","hsv78","hsv79","hsv80",
                                "hsv81")

# Add it to the labels data set
selectedLabels <- selectedLabels %>%
  inner_join(selectedFiles3,
             by = c("file" = "name"))

# See dimension of this data set
dim(selectedLabels)
```

```
[1] 250 167
```

**Question 5: Extract the number of lines**

The number of lines within an image can be extracted by using the hough_line() First, we pre-process the image using the cannyEdges() function before applying the hough_line() function [Note: The input should be a grayscale() image]. We use theta = 800 for the hough_line() function. From the output of the hough_line() function, we only want to retain the number of most important lines. Select those lines for which the score is in the .995th percentile or higher (use the quantile() function), and save this number of lines as an additional feature in the labels dataset

```{r}
# Convert the RGB images to grayscale using grayscale() in "imager" package
pictures4 <- lapply(pictures2,
                    FUN = function(f) grayscale(f))
```

```{r}
# Preprocess the image using cannyEdges()
pictures5 <- lapply(pictures4,
                    FUN = function(f) cannyEdges(f, alpha = 1))
```

```{r, warning=FALSE, message=FALSE}
# Create an empty matrix first. This matrix will store the number of lines for each picture later.
mat2 <- matrix(, nrow=0, ncol = 1)

for (picture in pictures5) {

  # Extract the lines
  lines <- hough_line(picture, ntheta = 800, data.frame = TRUE, shift = TRUE)

  # Extract the lines with score is in the 0.995th percentile or higher
  lines <- lines[lines$score >= quantile(lines$score, 0.995), ]

  # Extract the number of those lines and add that newly obtained number to the big matrix
  mat2 <- rbind(mat2, nrow(lines))
}

# Add the selected file names to the matrix so that we can merge it with the labels data set later
selectedFiles4 <- data.frame(cbind(selectedFiles2, mat2))
colnames(selectedFiles4) <- c("name", "lineNumber")
```

```
# Add it to the labels data set
selectedLabels <- selectedLabels %>%
  inner_join(selectedFiles4,
             by = c("file" = "name"))

# See dimension of this data set now
dim(selectedLabels)
```

```
[1] 250 168
```

**Question 6: Local Binary Pattern (LBP)**

Local Binary Pattern (LBP) is another texture feature we can extract. The function lbp() can do so, provided you apply it to a grayscale() transformed image. Note that applying grayscale() to an image creates a 4-dimensional array (check by applying dim()), of which you only need the first two dimensions that contain the image matrix. Create a histogram with 26 breaks from the LBP output using hist() on the ori argument of the output, and use the counts per cell as features in your labels dataset.

```{r}
# Check the dimension of a grayscale picture
dim(pictures4[[1]])

# Subset the first 2 dimentions of each grayscale picture
pictures6 <- lapply(pictures4,
                    FUN = function(f) f[1:150, 1:150])

# Check the dimension of a grayscale picture now
dim(pictures6[[1]])

# Create an empty matrix first. This matrix will store the counts of 26 breaks from the LBP output for each picture later.
mat3 <- matrix(, nrow=0, ncol = 26)
```

```
for (picture in pictures6) {

  # Extract the LBP
  lbp <- lbp(picture, r = 1)

  # Create a histogram with 26 breaks from the LBP output
  hist <- hist(lbp$lbp.ori, breaks = 26)

  # Extract the counts in each break and add those newly obtained numbers to the big matrix
  mat3 <- rbind(mat3, hist$count)
}
```

```
# Add the selected file names to the matrix so that we can merge it with the labels data set later
selectedFiles5 <- data.frame(cbind(selectedFiles2, mat3))

colnames(selectedFiles5) <- c("name",
                              "lbp1", "lbp2","lbp3","lbp4","lbp5","lbp6","lbp7","lbp8","lbp9","lbp10",
                              "lbp11", "lbp12","lbp13","lbp14","lbp15","lbp16","lbp17","lbp18","lbp19","lbp20",
                              "lbp21", "lbp22","lbp23","lbp24","lbp25","lbp26")


# Add it to the labels data set
selectedLabels <- selectedLabels %>%
  inner_join(selectedFiles5,
             by = c("file" = "name"))

# See dimension of this data set now
dim(selectedLabels)
```

**Question 7: Gabor filter**

R, 4 lines

Using the Gabor filter, we can extract texture of the image. The filter() function contains the means to do so. We will estimate B Gabor filters per image, using different combinations of lambda, theta and bw. We used lambda 2-8 with increments of 2, theta 0-180 with increments of 60, and bw 1-3 using increments of 1. One output of the function is the filtered_img. Take the mean and standard deviation of this filtered_img as your features for iteration B of the filter. Create a matrix with means and standard deviations of the B iterations of each image, and append these to the label dataset.

```
{r, warning = FALSE, message=FALSE}
# Make function to run Gabor filter for 48 (4x4x3) different combinations of lambda
, theta, and bw. Take mean and std deviation of filter_img and append to the list.
gabor_filter <- function(pic) {

  my_list <- vector(mode="list", length = 48)
  i <- 1

    for (lamda_ in seq(2, 8, 2)) {
      for (theta_ in seq(0, 180, 60)) {
        for (bw_ in seq(1, 3, 1)) {

          gabor <- gabor.filter(pic, lamda = lamda_, theta = theta_, bw = bw_, disp
= TRUE)
          my_list[i] <- mean(gabor$filtered_img)
          i <- i + 1
          my_list[i] <- sd(gabor$filtered_img)
          i <- i + 1
        }
      }
    }

  return(my_list)
  }
```

```
# Create empty matrix. This matrix will store the counts of 48 values from the
gabor_filter for each picture later.
mat4 <- matrix(, nrow=0, ncol = 96)

# Loop and run the function for all the images
for (picture in pictures6) {

  # Append the output to the empty matrix
  mat4 <- rbind(mat4, gabor_filter(picture) )
}
```

```r
# Add the selected file names to the matrix so that we can merge it with the labels
data set later
selectedFiles6 <- data.frame(cbind(selectedFiles2, mat4))

colnames(selectedFiles6) <- c("name", "mean1", "std1", "mean2", "std2", "mean3",
"std3", "mean4", "std4", "mean5", "std5", "mean6", "std6", "mean7", "std7", "mean8"
, "std8", "mean9", "std9", "mean10", "std10",
"mean11", "std11", "mean12", "std12", "mean13", "std13", "mean14", "std14",
"mean15", "std15", "mean16", "std16", "mean17", "std17", "mean18", "std18",
"mean19", "std19", "mean20", "std20",
"mean21", "std21", "mean22", "std22", "mean23", "std23", "mean24", "std24",
"mean25", "std25", "mean26", "std26", "mean27", "std27", "mean28", "std28",
"mean29", "std29", "mean30", "std30",
"mean31", "std31", "mean32", "std32", "mean33", "std33", "mean34", "std34",
"mean35", "std35", "mean36", "std36", "mean37", "std37", "mean38", "std38",
"mean39", "std39", "mean40", "std40",
"mean41", "std41", "mean42", "std42", "mean43", "std43", "mean44", "std44",
"mean45", "std45", "mean46", "std46", "mean47", "std47", "mean48", "std48")


selectedFiles6$name <- as.character(selectedFiles6$name)

# Add it to the labels data set
selectedLabels <- selectedLabels %>%
  inner_join(selectedFiles6,
             by = c("file" = "name"))
```

```r
# See dimension of this data set now
dim(selectedLabels)
```

# DATA MODELING

**Question 8: predictive models**

Having extracted the features, we can now develop some predictive models. Split the data into an 80% training set, and 20% holdout set. We will create two models, one to predict brand_label and one to predict product_label. You can make use of regular support vector machines (SVM) for this (included in library e1071 as the svm() function), but the L1-regularized L2-loss SVM is also available in library LiblineaR as the LiblineaR() function, with argument type = 5. Include all the extracted features as input, and evaluate the predictive performance of your models. Library MLmetrics contains functions for accuracy, recall, and F1 score.

```r
# convert labeled variables into factor variables
selectedLabels$product_label <- as.factor(selectedLabels$product_label)
selectedLabels$brand_label <- as.factor(selectedLabels$brand_label)
selectedLabels$product_name <- as.factor(selectedLabels$product_name)
selectedLabels$brand_name <- as.factor(selectedLabels$product_label)
selectedLabels$file <- as.factor(selectedLabels$file)

# convert all independent variables into numeric
selectedLabels[, -c(1:5)] <- sapply(selectedLabels[, -c(1:5)], as.numeric)

# feature scaling, for classification it's better to do feature scaling
additionally we have variables where the units are not the same
selectedLabels2 <- selectedLabels
selectedLabels2[, -c(1:5)] <- scale(selectedLabels2[, -c(1:5)])
```

```r
# train-test data split using caret library
set.seed(2350)
trainIndex <- createDataPartition(selectedLabels2$product_label, p = .8,
                                  list = FALSE,
                                  times = 1)
train <- selectedLabels2[ trainIndex,]
test <- selectedLabels2[-trainIndex,]

xTrainBrand <- subset(train, select = -brand_label)
yTrainBrand <- subset(train, select = brand_label)
xTestBrand <- subset(test, select = -brand_label)
yTestBrand <- subset(test, select = brand_label)

xTrainProduct <- subset(train, select = -product_label)
yTrainProduct <- subset(train, select = product_label)
xTestProduct <- subset(test, select = -product_label)
yTestProduct <- subset(test, select = product_label)
```

```r
# classification model for brand_label using e1071 library
modelBrand <- svm(brand_label ~ .-brand_name-product_name-product_label-file,
                  data = train,
                  type = 'C-classification',
                  kernel = 'radial')

# Make prediction
yPredBrand <- predict(modelBrand, xTestBrand)



# classification model for product_label using e1071 library
modelProduct <- svm(product_label ~ . -brand_name-brand_label-product_name-file,
                   data = train,
                   type = 'C-classification',
                   kernel = 'radial')

# Make prediction
yPredProduct <- predict(modelProduct, xTestProduct)
```

```{r}
# Precision for brand classification
precisionBrand <- Precision(yTestBrand$brand_label, yPredBrand, positive = NULL)
print(precisionBrand)

# Recall for brand classification
recallBrand <- Recall(yTestBrand$brand_label, yPredBrand, positive = NULL)
print(recallBrand)

# Accuracy for brand classification
accuracyBrand <- Accuracy(yTestBrand$brand_label, yPredBrand)
print(accuracyBrand)



# Precision for product classification
precisionProduct <- Precision(yTestProduct$product_label, yPredProduct, positive =
NULL)
print(precisionProduct)
```

```
# Precision for product classification
precisionProduct <- Precision(yTestProduct$product_label, yPredProduct, positive =
NULL)
print(precisionProduct)

# Recall for product classification
recallProduct <- Recall(yTestProduct$product_label, yPredProduct, positive = NULL)
print(recallProduct)

# Accuracy for product classification
accuracyProduct <- Accuracy(yTestProduct$product_label, yPredProduct)
print(accuracyProduct)
```

After using different kernels, we end up using RBF as the kernel for svm as it helps us obtain the highest precision, recall, and accuracy values in both models.

The model for brand classification is relatively better than product classification. The precision is high while the recall is lower, indicating that the model returns few positive results but most of its predicted labels are correct. The accuracy is higher than 0.5 which means this model performs better than random model.

Regarding product classification, the recall is high while the precision is low, indicating that the model returns many positive results, but most of its predicted labels are incorrect.

**End of work.** Thank you for your reading.