Using machine learning to predict basic shapes

Contents

1 Introduction

This report is part of the final course in the HarvardX Data Science Professional Certificate, Capstone, and it is available in GitHb: https://github.com/JeffersonMagalhaes/edx_basic_shapes. The aim of the project is applying machine learning techniques that go beyond standard linear regression. Therefore, we will create a model to classify images.

The dataset chosen is from https://www.kaggle.com/cactus3/basicshapes, and is a collection of 100 triangles, 100 squares and 100 circles. Each drawing is a png image 28x28 px. They are in 3 folders labeled squares, circles and triangles.

We will be creating our models using the tools we have learned throughout the courses. Thus, we will use R language to write the code, and its libraries, which will help us to complete the task. We will wrangle data, visualize it, and create a machine learning model.

First, we will analyse our dataset. Then we will create a train and a test set. After that, we will create a model that will be measured by its overall accuracy.

2 Analysis

2.1 Libraries

The first step is to define what libraries we are going to use. We will need tidyverse package to manipulate the dataset, caret package to develop our machine learning model, and OpenImageR to deal with the images.

```
knitr::opts_chunk$set(echo = TRUE)

############################################
#Loading packages
############################################

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ----------------------------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  2.0.1     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.3.1     v forcats 0.3.0
```

```
## -- Conflicts -------------------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(OpenImageR)) install.packages("OpenImageR", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: OpenImageR
```

```
if(!require(matrixStats)) install.packages("matrixStats", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: matrixStats
```

```
##
## Attaching package: 'matrixStats'

## The following object is masked from 'package:dplyr':
##
##     count
```

```
if(!require(stringi)) install.packages("stringi", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: stringi
```

```
if(!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-project.org")
```

2.2 Dataset

Since our dataset is a set o .png files, we need to convert it to a R object. The function readImage from OpenImageR reads a image a return a three dimensional matrix.

```
#shapes
shapes = c("circles","squares","triangles")

####read images to create a dataset of images
fig_dataset = lapply(shapes, function(x){
  file_names = paste(x,"\\",dir(x),sep="")
  read_files = lapply(file_names, function(y){
    fig = readImage(y)
    shape = x
    return(list(img = fig,shape = shape))
  })
})

class(fig_dataset)
```

```
## [1] "list"
```

```
###dataset is list of 3 list, one for each shape.
###It is better to have a unique list for all shapes, so will use unlist
fig_dataset = unlist(fig_dataset, recursive = F)

class(fig_dataset)
```

```
## [1] "list"
```

We can see that the length of our list is equal to 300 (100 triangles, 100 squares and 100 circles).

```
length(fig_dataset)
```

```
## [1] 300
```

Each is a list with a img object that represents our images, and shape field is the shape of the images.

```
names(fig_dataset[[1]])
```

```
## [1] "img"   "shape"
```

We can check that object of img is a array

```
class(fig_dataset[[1]]$img)
```

```
## [1] "array"
```

Each dimension of the array represents a matrix of RGB scale

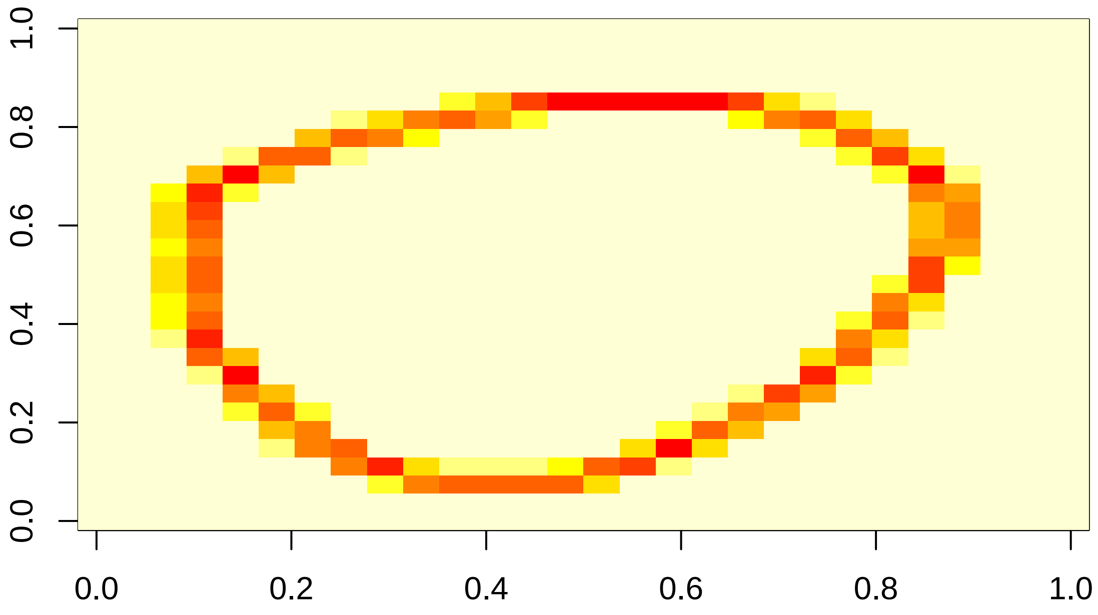
```
class(fig_dataset[[1]]$img[,,1])
```

```
## [1] "matrix"
```

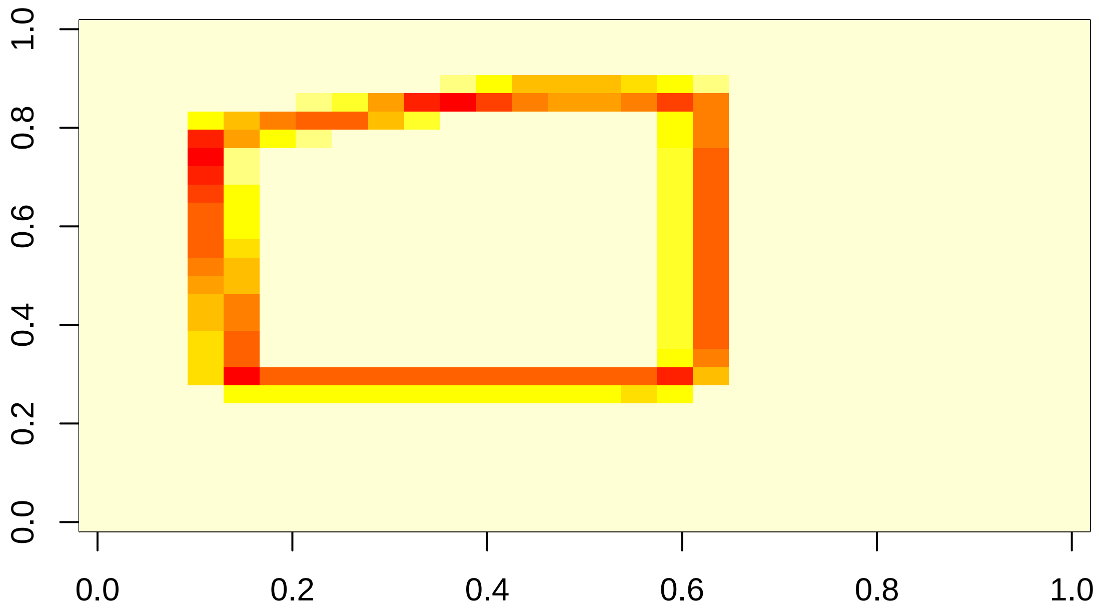
2.3 Shapes

The function image can be used to see some of the images.

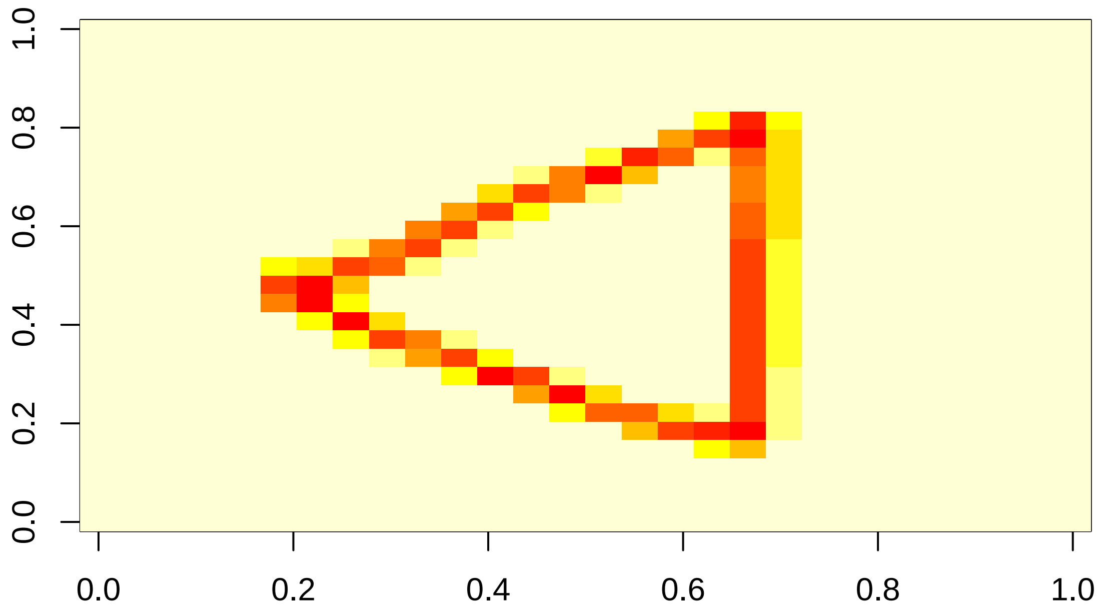
```
## [1] "circles"
```



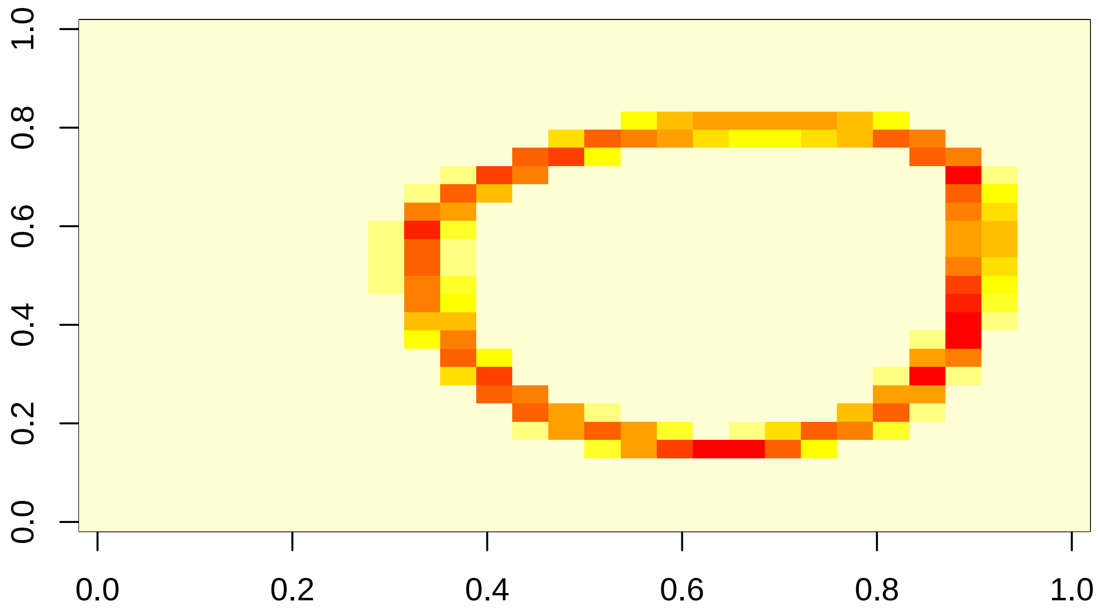
```
## [1] "squares"
```



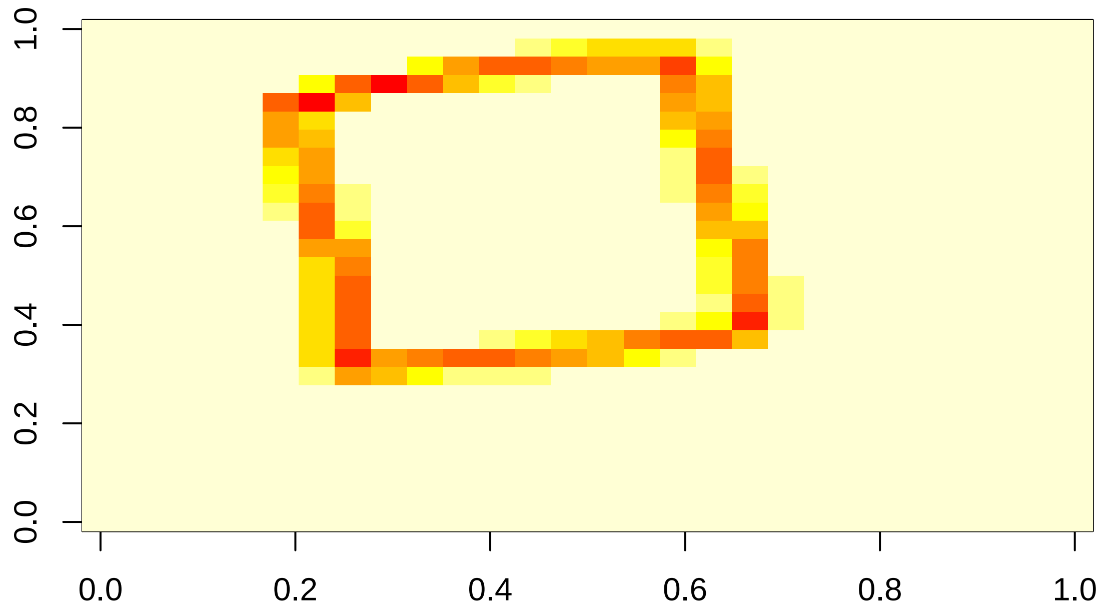
```
## [1] "triangles"
```



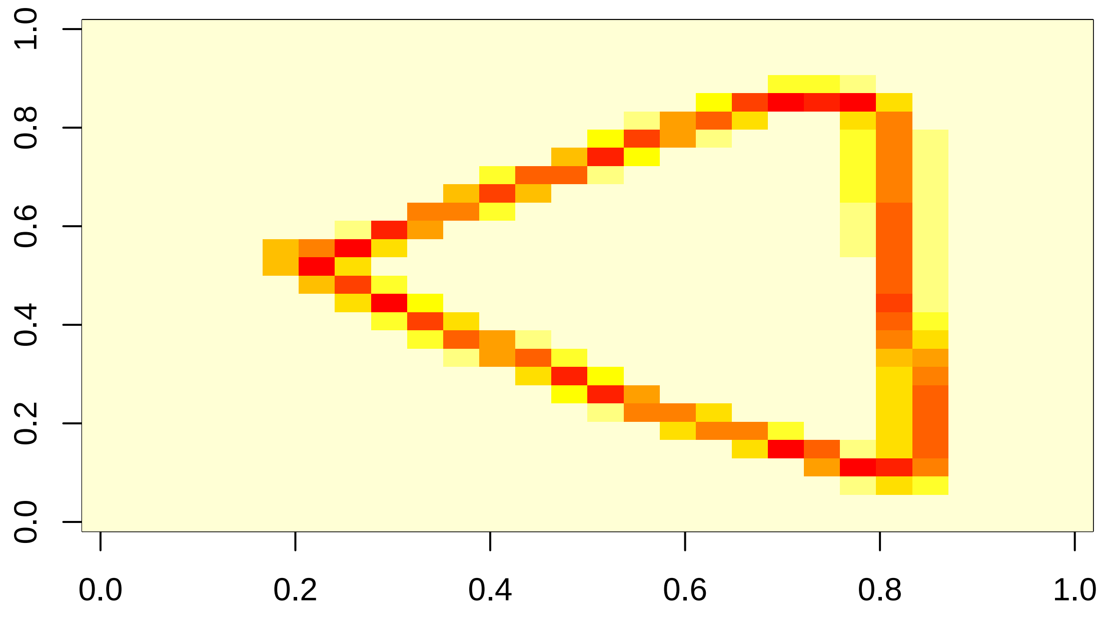
```
## [1] "circles"
```



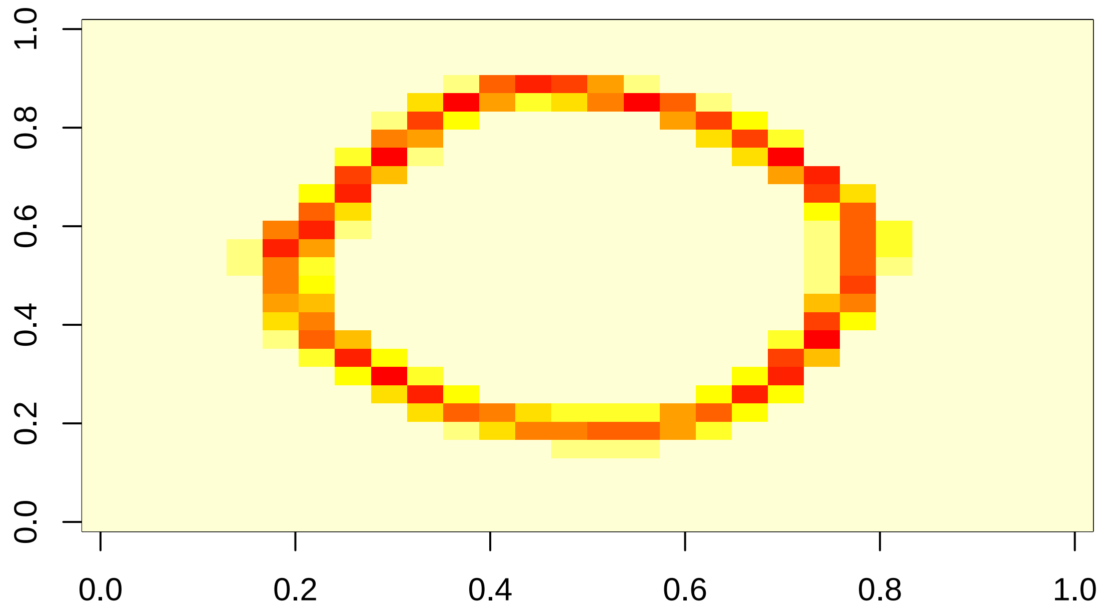
```
## [1] "squares"
```



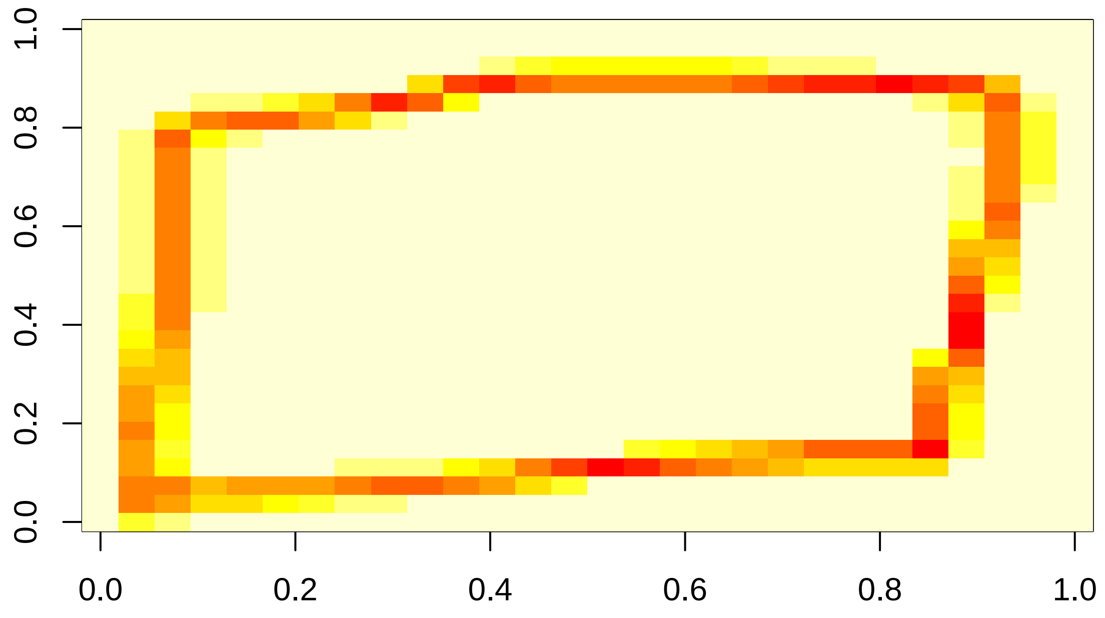
```
## [1] "triangles"
```



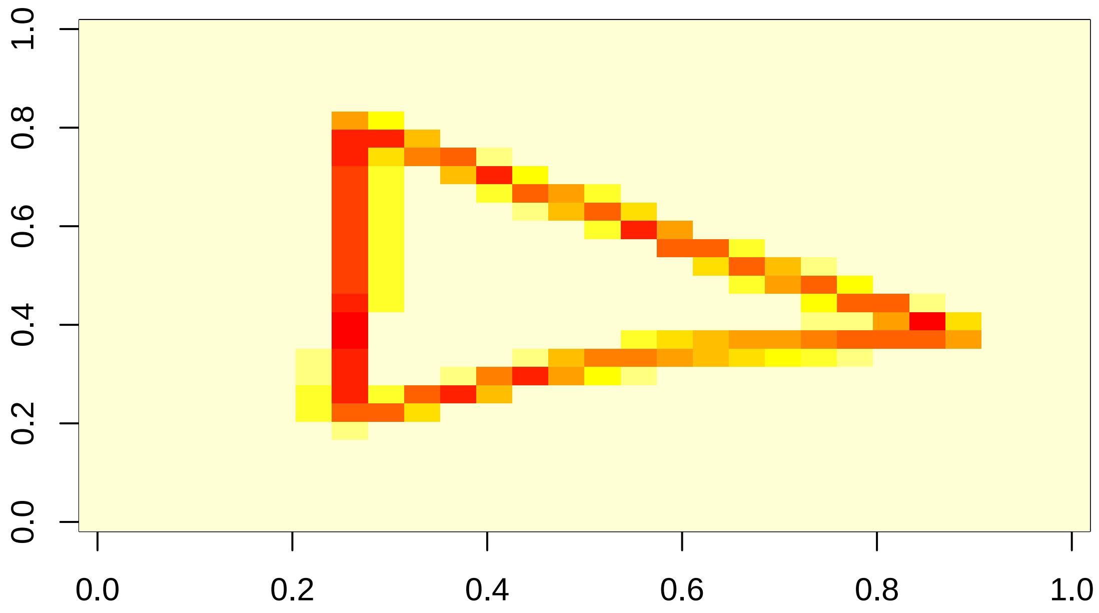
```
## [1] "circles"
```



```
## [1] "squares"
```



```
## [1] "triangles"
```



3 Methods

3.1 Train, and test sets

To create a machine will be needed to divide the dataset in a train and test sets. We will use 80% of the dataset to train our models (240 images), and other images will be used to test the models.

3.2 Gray scale

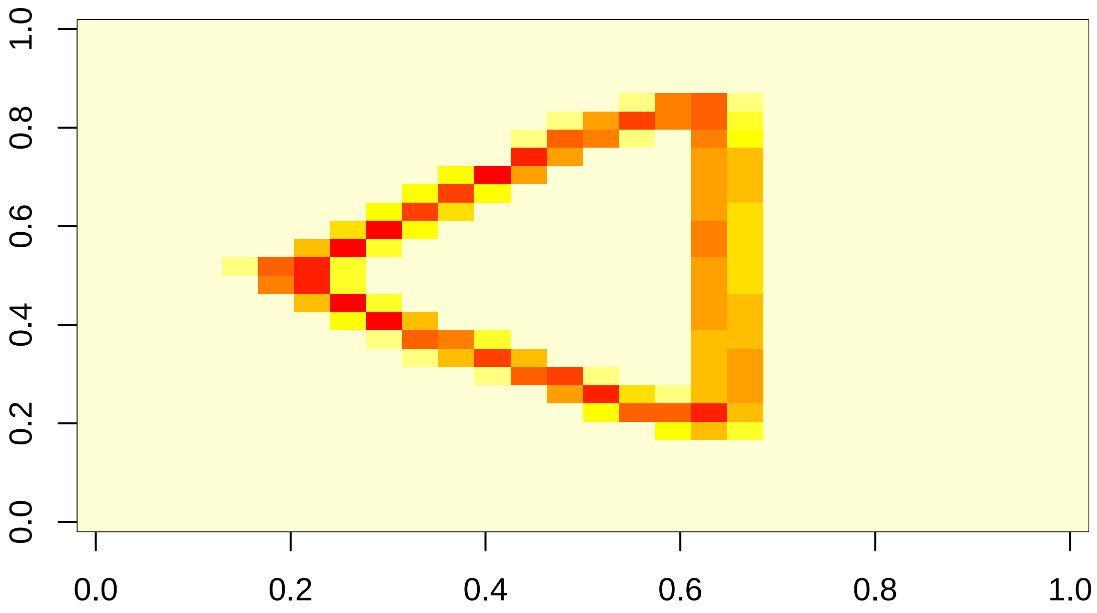
Because the RGB scale is not important to your problem, we are going to use the fucntion rgb_2gray from OpenImageR to transform our images to gray scale.

3.3 Data Augmentation

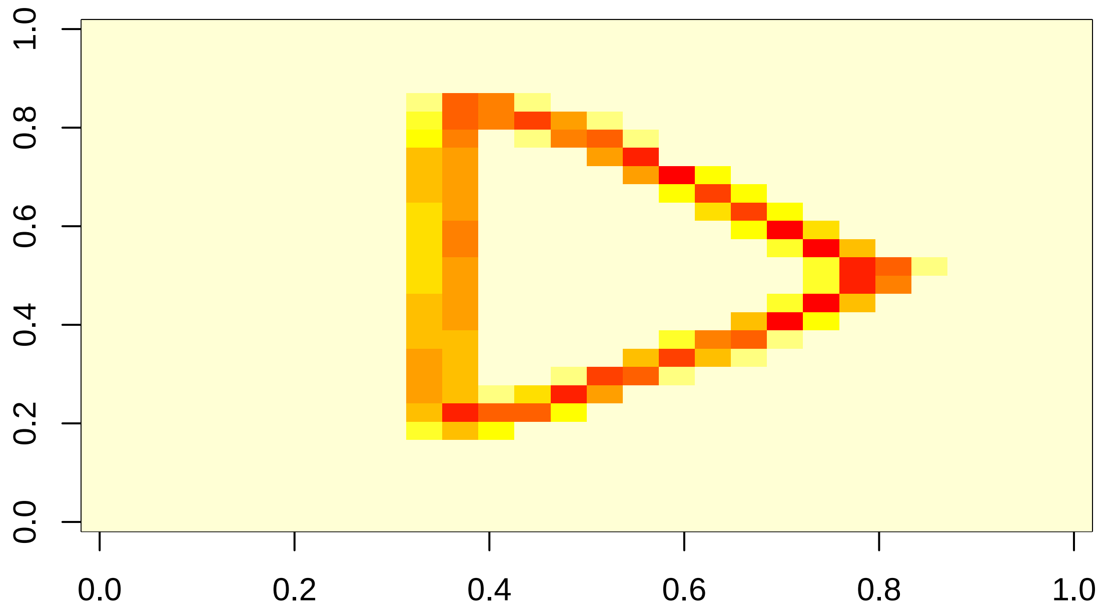
We will use data augmentation to increase our train set, since our it will have just 240 images, which is less than to total number of features $(28 * 28 = 784)$.

3.3.1 Flip

Fliping images doesn't change their shapes.



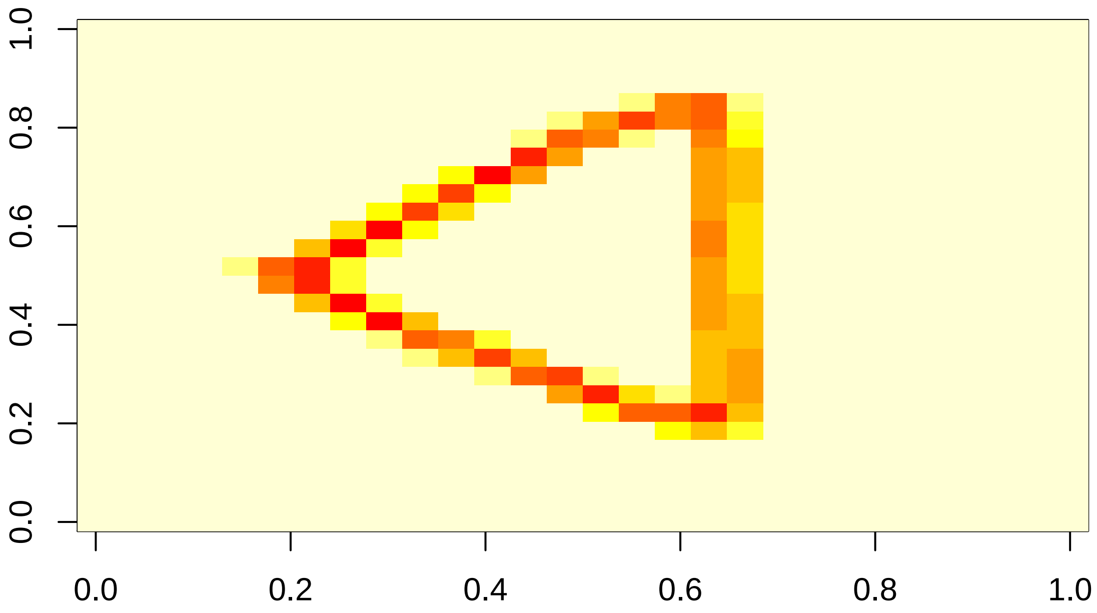
Original image



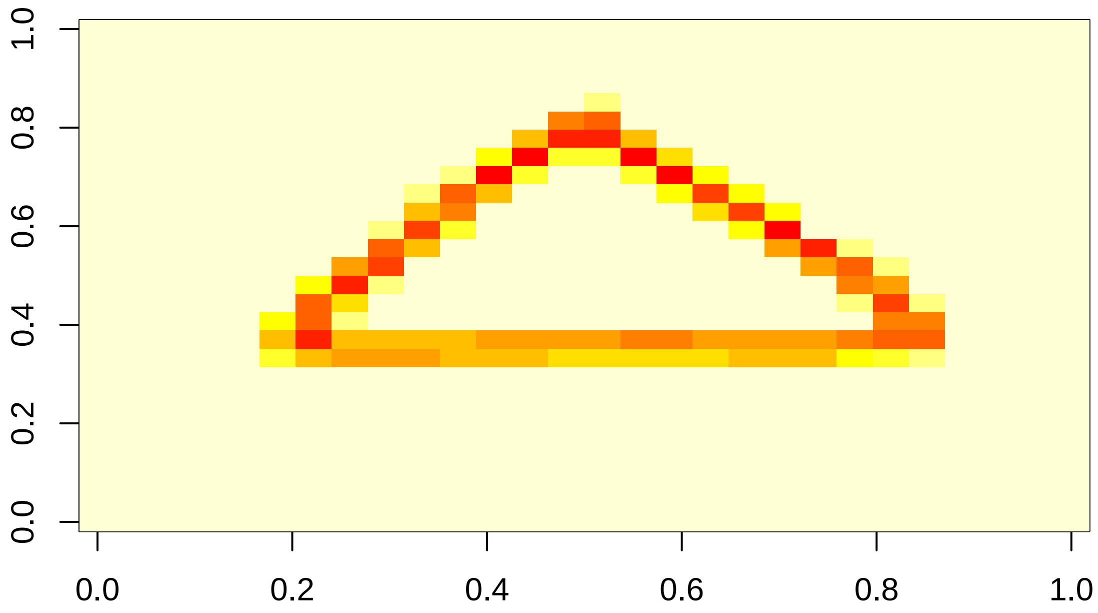
Flipped image

3.4 Rotate

Rotating images also doesn´t chage their shapes.



Original image



Rotated image

3.5 Histogram of Oriented Gradients (HOG)

We will use Histogram of Oriented Gradients. This technique decomposes an image into small squared cells, computes an histogram of oriented gradients in each cell, normalizes the result using a block-wise pattern, and return a descriptor for each cell. This is be completed by HOG fucntion from OpenImageR.

3.6 Models

We will train various models. Basically, we will use tree models, Support-vector machine (SVM) models, neural networking and some statistical models.

```
models <- c("lda",  "naive_bayes",  "svmLinear", "qda",
            "knn", "kknn", "loclda",
            "rf", "wsrf",
            "avNNet", "mlp", "monmlp","gbm",
            "svmRadial", "svmRadialCost", "svmRadialSigma")
```

3.7 Ensemble

Once we have trainned models, we will ensemble those that performe better.

4 Results

4.1 Train, and test sets

```
####because our dataset is a list and not a data.frame,
####we will first create a vector with the same size of our list
#### Then we will use it as a index to split our data

vec_img = 1:300
set.seed(1)
test_index = createDataPartition(vec_img,times = 1, p = 0.2, list = F)

test_set = fig_dataset[test_index]
train_set = fig_dataset[-test_index]
```

4.1.1 Train set

We will create a function to increase the train set by data augmentation and transform it by HOG.

```
###função para transformar em escala de cinza, augmentation e HOG (obtém features da figura)
GRAY_AUG_HOG = function(picture){
  pict = rgb_2gray(picture)
  img_flip_vert = HOG(flipImage(pict, mode = "vertical"))
  img_flip_hor = HOG(flipImage(pict, mode = "horizontal"))
  img_rot_90 = HOG(rotateFixed(pict, 90))
  img_rot_180 = HOG(rotateFixed(pict, 180))
  img_rot_270 = HOG(rotateFixed(pict, 270))
  img = HOG(pict)
  df = rbind(img,img_flip_vert,img_flip_hor,img_rot_90,img_rot_180,img_rot_270)
  return(df)
}
```

Then, we will use the function to reach our goal.

```
####Train dataset increased by Augmentation and transformed by HOG
train_df = lapply(train_set, function(x){
  fig = GRAY_AUG_HOG(x$img)
  data.frame(fig)%>% mutate(shape = x$shape)
})
train_df = plyr::ldply(train_df)
```

4.1.2 Test set

We will also transform our test set using HOG function.

```
test_df = lapply(test_set, function(x){
  fig = HOG(x$img)
  data.frame(t(fig),shape = x$shape)
})
test_df = plyr::ldply(test_df)
```

4.2 Trainning the models

We can use train fucntion from caret package to train and tuning our models.

```
fit = lapply(models, function(x){
  print(x)
  fit = train_df %>% train(shape~., data = ., method = x)
})
```

Then, we can see the overall accuracy of the model.

```
###accuracy
acc = sapply(fit, function(x){
  max(x$results$Accuracy, na.rm = T)
})
names(acc) = models
acc
```

```
##             lda     naive_bayes       svmLinear             qda             knn
##       0.7936935       0.7187268       0.7892512       0.8705978       0.7909726
##            kknn          loclda              rf            wsrf          avNNet
##       0.8294669       0.8887976       0.8799675       0.8815060       0.8094812
##             mlp          monmlp             gbm       svmRadial   svmRadialCost
##       0.7063273       0.7700051       0.8631464       0.8800285       0.8806162
## svmRadialSigma
##       0.8852369
```

4.3 Emsemble

Lastly, we can ensemble those models that performed better. We will keep those that their overall accucary on the train set was above 0.85.

```
model_acc =data.frame(model = models, acc = acc)
model_ok = model_acc %>% filter(acc > 0.85)
model_ok
```

model	acc
qda	0.8705978
loclda	0.8887976
rf	0.8799675
wsrf	0.8815060
gbm	0.8631464
svmRadial	0.8800285
svmRadialCost	0.8806162
svmRadialSigma	0.8852369

```
pred = predict(fit[acc>0.85])


pred = data.frame(matrix(unlist(pred), nrow = nrow(train_df)))
```

```
names(pred) = model_ok$model

pred_cl = rowSums(pred == "circles")
pred_sq = rowSums(pred == "squares")
pred_tr = rowSums(pred == "triangles")

preds = data.frame(pred, circle = pred_cl, square = pred_sq, triangle = pred_tr) %>%
  mutate(pred = ifelse(circle>square & circle>triangle,"circles",
                     ifelse(square>circle & square>triangle,"squares",
                            ifelse(triangle>circle &triangle>square, "triangles",
                                   as.character(wsrf)))))

confusionMatrix(as.factor(preds$pred), as.factor(train_df$shape))
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction  circles squares triangles
##   circles       455       7         0
##   squares        18     473         8
##   triangles       1       0       478
##
## Overall Statistics
##
##                Accuracy : 0.9764
##                  95% CI : (0.9672, 0.9836)
##     No Information Rate : 0.3375
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9646
##  Mcnemar's Test P-Value : 0.003131
##
## Statistics by Class:
##
##                      Class: circles Class: squares Class: triangles
## Sensitivity                  0.9599         0.9854           0.9835
## Specificity                  0.9928         0.9729           0.9990
## Pos Pred Value               0.9848         0.9479           0.9979
## Neg Pred Value               0.9806         0.9926           0.9917
## Prevalence                   0.3292         0.3333           0.3375
## Detection Rate               0.3160         0.3285           0.3319
## Detection Prevalence         0.3208         0.3465           0.3326
## Balanced Accuracy            0.9763         0.9792           0.9912
```

4.4 Overall Accuracy on the test set

Once we have created our model, we can test it on the test set.

```
pred_test = predict(fit[acc>0.85], newdata = test_df)
pred_test = data.frame(matrix(unlist(pred_test), nrow = nrow(test_df)))
names(pred_test) = model_ok$model
```

```
pred_cl_test = rowSums(pred_test == "circles")
pred_sq_test = rowSums(pred_test == "squares")
pred_tr_test = rowSums(pred_test == "triangles")

preds_test = data.frame(pred_test, circle = pred_cl_test, square = pred_sq_test, triangle = pred_tr_tes
  mutate(pred = ifelse(circle>square & circle>triangle,"circles",
                      ifelse(square>circle & square>triangle,"squares",
                             ifelse(triangle>circle &triangle>square, "triangles",
                                    as.character(wsrf)))))
```

We can see that the overall accuracy is 0.9167.

```
CMpred = confusionMatrix(as.factor(preds_test$pred), as.factor(test_df$shape))
CMpred$overall[[1]]
```

```
## [1] 0.9166667
```

5 Conclusion

In this project, we have created a model to predict basic shapes. We have used various models, such as loclda, and svmRadialSigma, and the final model is a combination of those that performed better on the train set. The final overall accurracy is 0.9167. Since it is above 0.90, we consider that we have reached our goal. Although we have end with a good model, we recommend increase the dataset for future projects. This also helps to create the learning curve, each determines how the model is improving.