Data representations for neural networks

Practice

Contents

```
library(keras)
library(stringr)
library(class)
library(ggplot2)
library(ggseqlogo)
```

```
## Warning: package 'ggseqlogo' was built under R version 4.0.5
```

```
library(OpenImageR)
```

Example 1. Twiter

```
dat<-read.csv("sa.csv",header=F, sep=";")
str(dat)
```

```
## 'data.frame':    2000 obs. of  2 variables:
##  $ V1: chr  "Wow... Loved this place." "Crust is not good." "Not tasty and the texture was just nasty
##  $ V2: int  1 0 0 1 1 0 0 0 1 1 ...
```

```
texts<-dat[,1]
labels<-dat[,2]
```

Encoding

```
maxlen <- 25
max_words <- 5000  # to only take into account the 5000 most common words
tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)   # builds the word index
sequences <- texts_to_sequences(tokenizer, texts)  # turns strings into lists of integers indices
word_index = tokenizer$word_index # how you can recover the word index that was computed
```

```
texts[1:5]
```

```
## [1] "Wow... Loved this place."
## [2] "Crust is not good."
## [3] "Not tasty and the texture was just nasty."
## [4] "Stopped by during the late May bank holiday off Rick Steve recommendation and loved it."
## [5] "The selection on the menu was great and so were the prices."
```

```
names(word_index)[1:100]
```

```
##   [1] "the"       "and"       "i"         "a"         "it"        "to"
##   [7] "is"        "was"       "this"      "of"        "not"       "for"      
##  [13] "my"        "in"        "with"      "very"      "good"      "great"    
##  [19] "phone"     "that"      "on"        "have"      "you"       "food"     
##  [25] "had"       "place"     "so"        "service"   "but"       "are"      
##  [31] "be"        "we"        "all"       "as"        "at"        "like"     
##  [37] "they"      "time"      "back"      "were"      "one"       "quality"  
##  [43] "would"     "really"    "here"      "if"        "your"      "well"     
##  [49] "from"      "just"      "product"   "up"        "don't"     "best"     
##  [55] "no"        "will"      "an"        "go"        "there"     "only"     
##  [61] "also"      "has"       "me"        "out"       "i've"      "works"    
##  [67] "ever"      "nice"      "headset"   "battery"   "it's"      "sound"    
##  [73] "than"      "use"       "our"       "when"      "i'm"       "or"       
##  [79] "what"      "get"       "been"      "after"     "their"     "love"     
##  [85] "did"       "excellent" "even"      "recommend" "too"       "which"    
##  [91] "first"     "again"     "more"      "work"      "about"     "better"   
##  [97] "ear"       "never"     "2"         "price"
```

```
sequences[1:5]
```

```
## [[1]]
## [1] 507 268   9  26
## 
## [[2]]
## [1] 967   7  11  17
## 
## [[3]]
## [1]  11 269   2   1 730   8  50 731
## 
## [[4]]
##  [1]  732  112  396    1  968  354 1427 1428  136 1429 1430  733    2  268    5
## 
## [[5]]
##  [1]   1 291  21   1 195   8  18   2  27  40   1 292
```

```
cat("Found", length(word_index), "unique tokens.\n")
```

```
## Found 3258 unique tokens.
```

```
data <- pad_sequences(sequences, maxlen = maxlen)
# turns the lists of integers into a 2D integer tensor of shape (samples, maxlen)
dim(data)
```

```
## [1] 2000   25
```

```
data[5,]
```

```
##  [1]   0   0   0   0   0   0   0   0   0   0   0   0   0   1 291  21   1 195   8
## [20]  18   2  27  40   1 292
```

```
labels <- as.array(labels)
labels[1:100]
```

```
##   [1] 1 0 0 1 1 0 0 0 1 1 1 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 0 1 0 0 1 0 1 0 1 1 1
##  [38] 0 1 0 1 0 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 0
##  [75] 0 0 0 1 1 0 0 0 0 1 0 1 0 1 1 1 0 1 0 1 0 0 1 1 0 1
```

```
cat("Shape of data tensor:", dim(data), "\n")
```

```
## Shape of data tensor: 2000 25
```

```r
cat('Shape of label tensor:', dim(labels), "\n")
```

```
## Shape of label tensor: 2000
```

Split in train and test set

```r
set.seed(123)
p<-0.75
training_indices <-sample(nrow(data),p*nrow(data))
x_train <- data[training_indices,]
y_train <- labels[training_indices]
x_test <- data[-training_indices,]
y_test <- labels[-training_indices]
```

Example 2. Splicing

Splice junctions are points in a DNA sequence where "superfluous" DNA is removed during the process of protein synthesis in higher organisms. The problem with this dataset is to recognize, given a DNA sequence, the boundaries between exons (the parts of the DNA sequence retained after *splicing*) and introns (the parts of the DNA sequence retained after *splicing*) that are cut).

This problem consists of two subtasks: recognizing exon/intron boundaries (called EI sites) and recognizing intron/exon boundaries (IE sites). In the biological community, IE boundaries are called `acceptors`, while IE boundaries are called `donors`. All examples were taken from Genbank 64.1. The "EI" and "IE" categories include primate *spliced genes* in Genbank 64.1. Examples of no *splcing* were taken from sequences known not to include a *splicing* site.

The data is available in the `splice.txt` dataset. The file contains 3190 rows that correspond to the different sequences, and 3 columns separated by commas. The first corresponding column to the class of the sequence (ei, ie or n), the second column with the identifying name of the sequence and the third column with the sequence itself. In the case of DNA sequences, the nucleotides will appear identified in a standard way with the letters A, G, T and C. In addition, other characters appear between the characters standard, D, N, S and R, which indicate ambiguity according to the following table:

```
|code     | nucleotides       |
|---------|-------------------|
|D        | A or G or T       |
|N        | A or G or C or T  |
|S        | C or G            |
|R        | A or3 G           |
```

For more information about the case, you can consult the link: https://archive.ics.uci.edu/ml/datasets/Molecular+Biology+(Splice-junction+Gene+Sequences), from the University of California-Irvine (UCI) machine learning repository.

The way chosen to represent the data is a crucial step in the algorithms. In the case at hand, sequence-based analysis, one-hot coding will be used.

The `one-hot` coding represents each nucleotide by a vector of 8 components, with 7 of them at 0 and one at 1. Say for example, nucleotide A is represented by (1,0,0,0,0,0,0,0), nucleotide G by (0,1,0,0,0,0,0,0), the T by (0,0,1,0,0,0,0,0) and, finally, the C by (0,0,0,1,0,0,0,0) and the ambiguity characters will be represented, the D for (0,0,0,0,1,0,0,0), the N for (0,0,0,0,0,1,0,0), the S for (0,0, 0,0,0,0,1,0) and the R for (0,0,0,0,0,0,0,1).

So each sequence of 60 nucleotides will become a vector of 8*60=480 components resulting from concatenating the vectors for each of the 60 nucleotides. As an example, the first sequence of the database and the result of the one-hot encoding are shown. Note that it will be necessary to remove the blank spaces before the sequences. We can use the `str_trim` function from the stringr package.

```r
splice <- read.csv("splice.data", header=FALSE)


# A,G,T,C,D,N,S,R

A<-c(1,0,0,0,0,0,0,0)
G<-c(0,1,0,0,0,0,0,0)
T<-c(0,0,1,0,0,0,0,0)
C<-c(0,0,0,1,0,0,0,0)
D<-c(0,0,0,0,1,0,0,0)
N<-c(0,0,0,0,0,1,0,0)
S<-c(0,0,0,0,0,0,1,0)
R<-c(0,0,0,0,0,0,0,1)


df<-data.frame()

for( s in 1:100){ # 3190

  seq<-unlist(strsplit(str_trim(splice[s,3]),split=NULL))
  v<-vector()

  for (i in 1:length(seq)){

    xchar<-seq[i]

    if (xchar=="A") v<-c(v,A)
    if (xchar=="G") v<-c(v,G)
    if (xchar=="T") v<-c(v,T)
    if (xchar=="C") v<-c(v,C)
    if (xchar=="D") v<-c(v,D)
    if (xchar=="N") v<-c(v,N)
    if (xchar=="S") v<-c(v,S)
    if (xchar=="R") v<-c(v,R)

  }
  df[s,1:480]<-v    # 480=60*8
}
```

```r
class(df)
```

```
## [1] "data.frame"
```

```r
df1 <- as.matrix(df)
dim(df1)
```

```
## [1] 100 480
```

```r
df2<-array_reshape(df1,dim=c(nrow(df1),480,1) ) # conv-1d
dim(df2)
```

```
## [1] 100 480   1
```

```r
df3<-array_reshape(df1,dim=c(nrow(df1),60,8))    # conv-2d
dim(df3)
```

```
## [1] 100  60   8
```

```r
df2[1,,]
```

```
##   [1] 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0
##  [38] 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
##  [75] 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0
## [112] 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0
## [149] 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
## [186] 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
## [223] 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
## [260] 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
## [297] 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0
## [334] 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
## [371] 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0
## [408] 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0
## [445] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0
```

```r
df3[1,,]
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
##  [1,]    0    0    0    1    0    0    0    0
##  [2,]    0    0    0    1    0    0    0    0
##  [3,]    1    0    0    0    0    0    0    0
##  [4,]    0    1    0    0    0    0    0    0
##  [5,]    0    0    0    1    0    0    0    0
##  [6,]    0    0    1    0    0    0    0    0
##  [7,]    0    1    0    0    0    0    0    0
##  [8,]    0    0    0    1    0    0    0    0
##  [9,]    1    0    0    0    0    0    0    0
## [10,]    0    0    1    0    0    0    0    0
## [11,]    0    0    0    1    0    0    0    0
## [12,]    1    0    0    0    0    0    0    0
## [13,]    0    0    0    1    0    0    0    0
## [14,]    1    0    0    0    0    0    0    0
## [15,]    0    1    0    0    0    0    0    0
## [16,]    0    1    0    0    0    0    0    0
## [17,]    1    0    0    0    0    0    0    0
## [18,]    0    1    0    0    0    0    0    0
## [19,]    0    1    0    0    0    0    0    0
## [20,]    0    0    0    1    0    0    0    0
## [21,]    0    0    0    1    0    0    0    0
## [22,]    1    0    0    0    0    0    0    0
## [23,]    0    1    0    0    0    0    0    0
## [24,]    0    0    0    1    0    0    0    0
## [25,]    0    1    0    0    0    0    0    0
## [26,]    1    0    0    0    0    0    0    0
## [27,]    0    1    0    0    0    0    0    0
## [28,]    0    0    0    1    0    0    0    0
## [29,]    1    0    0    0    0    0    0    0
## [30,]    0    1    0    0    0    0    0    0
## [31,]    0    1    0    0    0    0    0    0
## [32,]    0    0    1    0    0    0    0    0
## [33,]    0    0    0    1    0    0    0    0
## [34,]    0    0    1    0    0    0    0    0
## [35,]    0    1    0    0    0    0    0    0
## [36,]    0    0    1    0    0    0    0    0
## [37,]    0    0    1    0    0    0    0    0
## [38,]    0    0    0    1    0    0    0    0
## [39,]    0    0    0    1    0    0    0    0
## [40,]    1    0    0    0    0    0    0    0
## [41,]    1    0    0    0    0    0    0    0
## [42,]    0    1    0    0    0    0    0    0
## [43,]    0    1    0    0    0    0    0    0
## [44,]    0    1    0    0    0    0    0    0
## [45,]    0    0    0    1    0    0    0    0
## [46,]    0    0    0    1    0    0    0    0
## [47,]    0    0    1    0    0    0    0    0
## [48,]    0    0    1    0    0    0    0    0
## [49,]    0    0    0    1    0    0    0    0
## [50,]    0    1    0    0    0    0    0    0
## [51,]    1    0    0    0    0    0    0    0
## [52,]    0    1    0    0    0    0    0    0
## [53,]    0    0    0    1    0    0    0    0
## [54,]    0    0    0    1    0    0    0    0
## [55,]    1    0    0    0    0    0    0    0
## [56,]    0    1    0    0    0    0    0    0
## [57,]    0    0    1    0    0    0    0    0
## [58,]    0    0    0    1    0    0    0    0
## [59,]    0    0    1    0    0    0    0    0
## [60,]    0    1    0    0    0    0    0    0
```

```r
set.seed(123)
train<-sample(1:nrow(df3),nrow(df3)*2/3)

trainset<-df3[train,,]
dim(trainset)
```

```
## [1] 66 60  8
```

```r
testset<-df3[-train,,]
dim(testset)
```

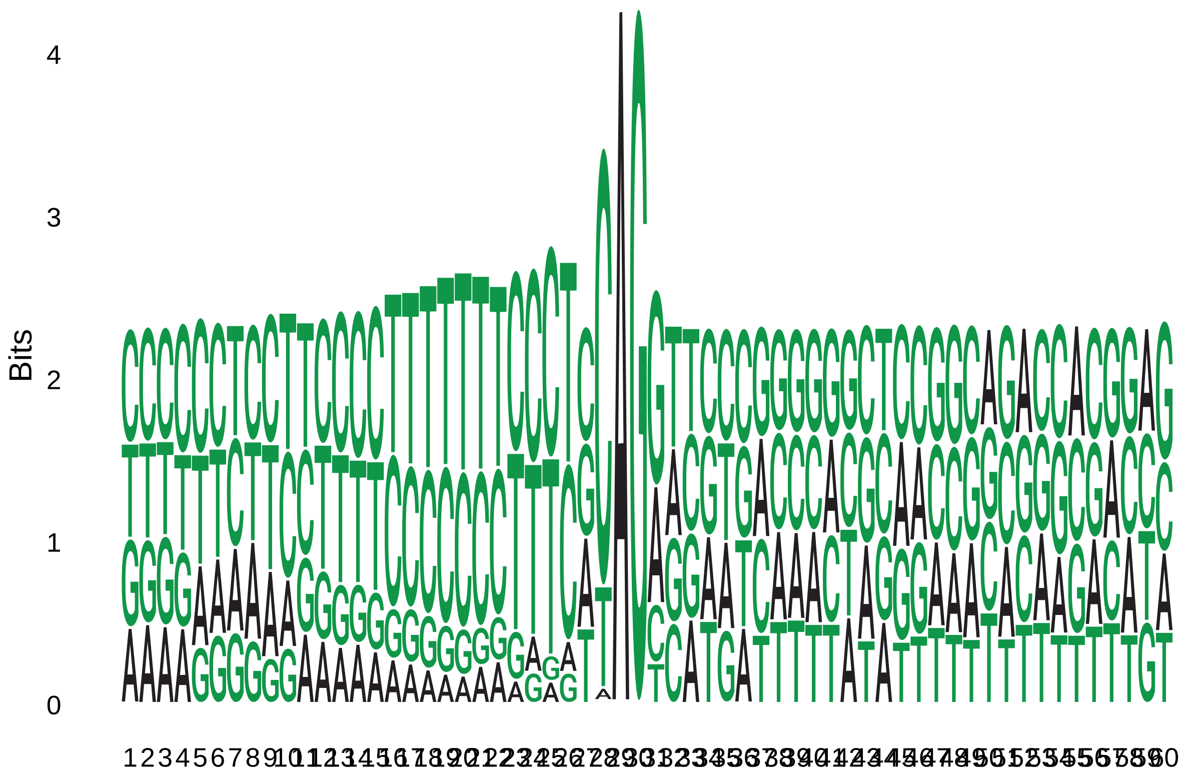
```
## [1] 34 60  8
```

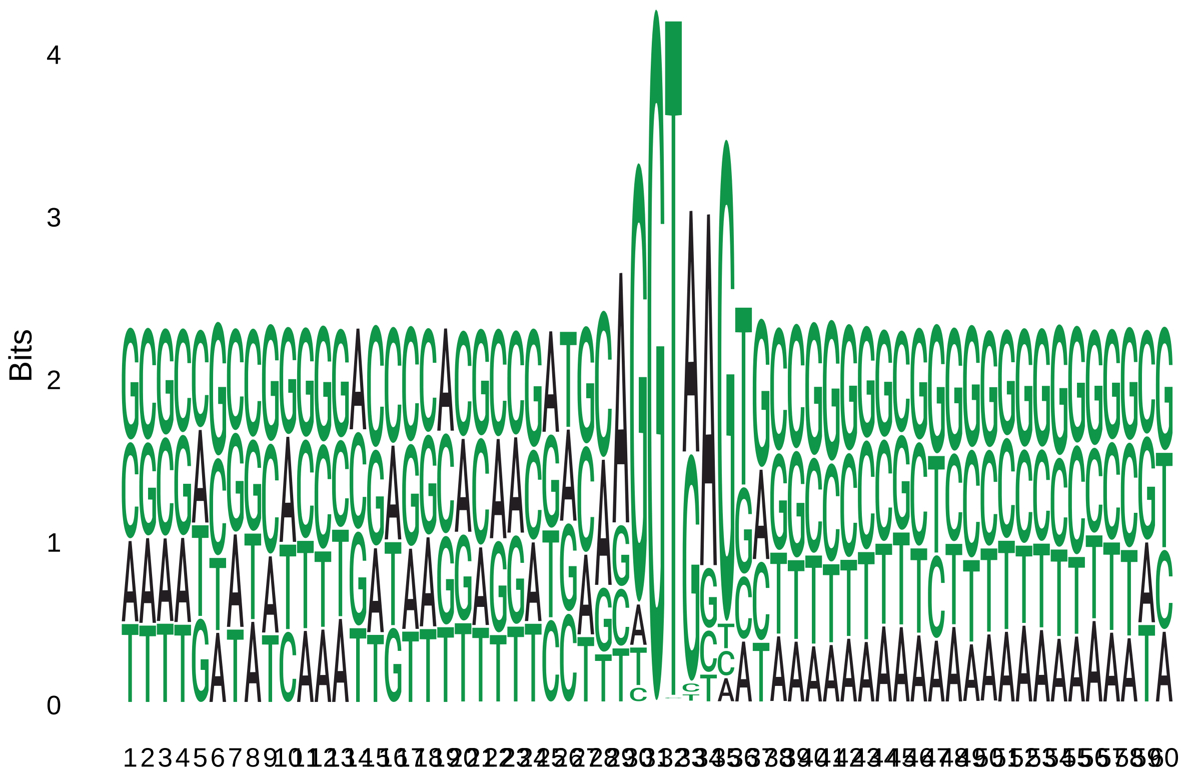
Logo sequences

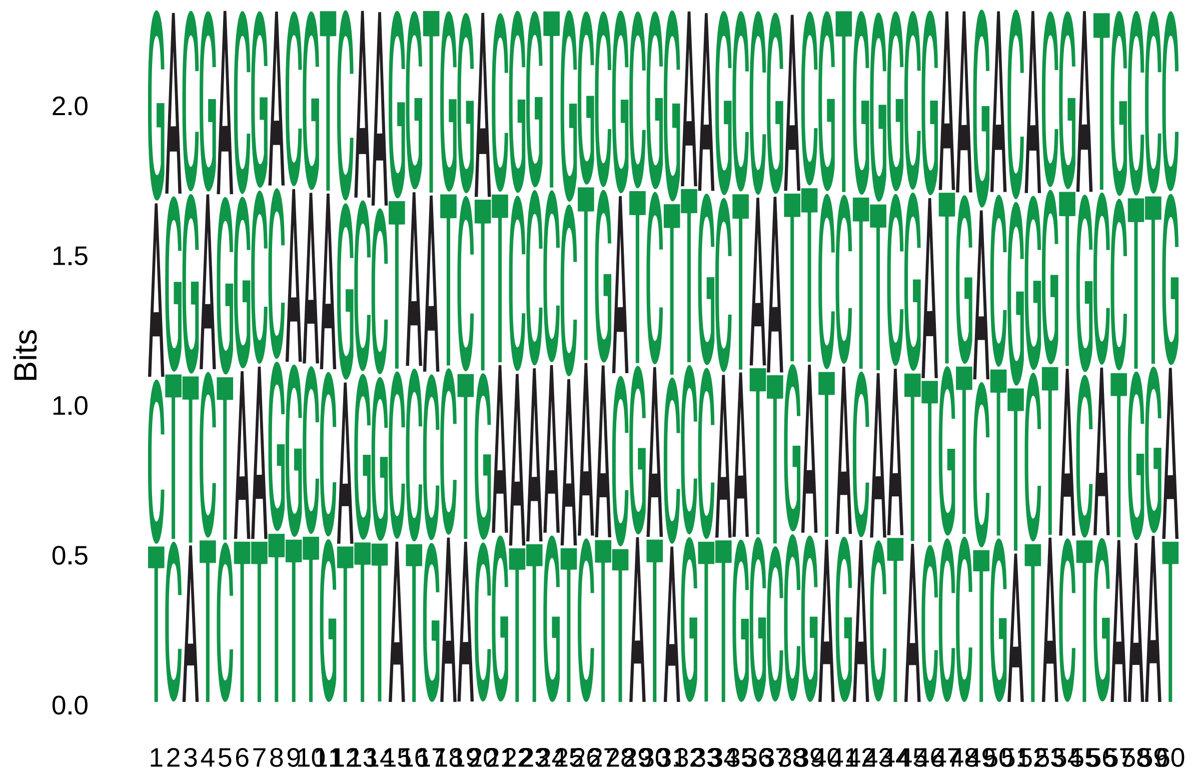
One way to present the sequence pattern graphically is to make a logo sequence (see https://en.wikipedia.org/wiki/Sequence_logo)).

As an example, we show you the logo sequence of the class "EI"

To create sequence logos, related sequences of DNA, RNA, or protein, or sequences of DNAs that share conserved binding sites are aligned until the most conserved parts create good alignments. A sequence logo can then be created from the multiple alignment of conserved sequences. The sequence logo will show the degree of conservation of the residues in each position: a lower number of different residues will cause larger letters, since the conservation is better in that position. Different residuals at the same position will be scaled according to their frequency. Sequence logos can be used to represent conserved DNA-binding sites, where the transcription factors remain attached (extracted from https://es.wikipedia.org/wiki/Logo_de_secuencias). To perform this representation, the ggseqlogo package downloadable from CRAN is used.


Bits
0
1
2
3
4


Bits
0
1
2
3
4



Example 3. Image analysis

```
########################  FACES ##############################
lf<-list.files(path="~/Docencia/curs21_22/UB/MESIO/DL/unitat2/tensor_types/faces32x32")
x_train<-array(dim=c(100,32,32,3))
for(i in 1:100){
  x_train[i,,,]<-readImage(paste0("~/Docencia/curs21_22/UB/MESIO/DL/unitat2/tensor_types/faces32x32/",l
}
```

```
img<-x_train[1,,,]
imageShow(img)
```

```r
i<-1000
img<-image_load(paste0("~/Docencia/curs21_22/UB/MESIO/DL/unitat2/tensor_types/faces32x32/",lf[i]))
img_tensor<-image_to_array(img)
dim(img_tensor)
```

```
## [1] 32 32  3
```

```r
img_tensor<-img_tensor/255
plot(as.raster((img_tensor)))
```