Artificial Neural Networks & Deep Learning

Ignasi Mane, Antoni Company & Chiara Barbi

2/6/2019

Data set: Caltech 101 Description.

In the web page: http://www.vision.caltech.edu/Image_Datasets/Caltech101/ there are pictures of objects belonging to 101 categories. About 40 to 800 images per category. Most categories have about 50 images. Collected in September 2003 by Fei-Fei Li, Marco Andreetto, and Marc Aurelio Ranzato. The size of each image is roughly 300 x 200 pixels.

1. NN Architectures.

1. Select five categories from caltech101 collection. Try to select balanced categories with representative number of images.
2. Implements a classifier fed with the HOG descriptors that learns the category to which a given image belongs. In HOG use 3 cells and 9 orientations. Compare the performance of two NN architectures:

2.1. Layer 1: 81 units (input layer), Layer 2: 10 units (hidden layer), Layer 3: 5 units (output layer).

2.2. Layer 1: 81 units (input layer), Layer 2: 50 units (hidden layer), Layer 3: 25 units (hidden layer), Layer 4: 5 units (output layer).

Question 1. Reshape grayscale images to 32x32. Vectorize images by rows, to get a 1024-vector representation of each image. Implement a NN with two hidden layers that uses such a vectors as input data for learning image categories. Tune the number of neurons in each hidden layer using a grid search procedure.

```
library(OpenImageR)
library(caret)
library(keras)
library(Xmisc)
library(ggplot2)
library(RColorBrewer)
library(ggrepel)
library(Rtsne)
set.seed(1234) #seed for the assignment
```

Importing data.

First we have defined the five categories selected: bass, crocodile, dolphin, elephant and llama.

For this exercise we will only use the first 30 images from each category.

Once the five categories are defined, we will define the path to upload the images from each category. First we define the folder path to each category:

```
names <- c("bass", "crocodile", "dolphin", "elephant", "llama")
root <- "101_ObjectCategories/"
path <- c()
for (i in names){
  path <- c(path, paste(root,i, sep = ""))
}
```

Then, we proceed with the picture path.

```
file.path <- c()
for (i in path){
  for (files in (list.files(i)[1:50])){
    file.path <- c(file.path, paste(i, files, sep = "/"))
  }
}
```

Once the paths are established, we will proceed importing the images to transform them into grey and to size 32x32 with a vector format.

```
m <-  1
output <- matrix(0, nrow = length(file.path), ncol = 1024)

for (i in file.path){
  image <- readImage(i)
  if (is.na(dim(image)[3]) == FALSE){
    image.grey <- rgb_2gray(image)}
  else{
    image.grey <- image
    }

  img.resh <- resizeImage(image.grey, width = 32,
                                height = 32, method = 'bilinear') #reshape
  img.vector <- as.vector(img.resh) # as 1024 vector
  output[m, ] <- img.vector
  m <- m + 1
}
```

Then, we will normalize the vector of images with values between 0 and 1 to use for the neural network.

```
# Normalization of the image information for the NN
normalizer <- function(x) {
    return((x - min(x)) / (max(x) - min(x)))
    }
x_train <- as.data.frame(lapply(as.data.frame(output), normalizer))
```

Once the images are uploaded for the neural network, we will continue importing the images with HOG.

```
o.hog <- matrix(0, nrow = length(file.path), ncol = 81)
m <- 1
for (a in file.path){
  image.hog <- HOG(readImage(file.path(a)), cells = 3, orientations = 9)
  o.hog[m, ] <- image.hog
  m <- m + 1
}
```

Finally, we will define the five categories of the images.

```
y.res <- array(c(rep(c(0,1,2,3,4), each = 30))) %>%
            array(dim = c(length(file.path))) %>%
            to_categorical(num_classes = 5)
```

Neural Network training.

We will start with the training of the neural network by tuning the number of hidden layers using the grid search procedure.

```
res.grid <- data.frame()
m <- 1
for (firstlayer in c(25, 80, 100)){
  for (secondlayer in c(10, 50, 70)){

    start_time <- Sys.time()
    model <- keras_model_sequential()
    model %>%
        layer_dense(units = dim(x_train)[1], activation = 'relu',
                    input_shape = dim(x_train)[2]) %>%
        layer_dense(units = firstlayer, activation = 'relu') %>%
        layer_dense(units = secondlayer, activation = 'relu') %>%
        layer_dense(units = 5, activation = 'softmax') %>%
        compile(
                optimizer = 'rmsprop',
                loss = 'categorical_crossentropy',
                metrics = c('accuracy')
                )
    out <- model %>% fit(as.matrix(x_train), y.res, epochs = 100)
    end_time <- Sys.time()


    res.grid[m, 1] <- firstlayer
    res.grid[m, 2] <- secondlayer
    res.grid[m, 3] <- out$metrics$acc[length(out$metrics$acc)]
    res.grid[m, 4] <- end_time - start_time
    m <- m + 1
  }
   }

colnames(res.grid) <- c("Layer1","Layer2","Accuracy","Time")
```

The results obtained are:

```
res.grid
```

```
##   Layer1 Layer2 Accuracy          Time
## 1     25     10    0.800 5.366263 secs
## 2     25     50    0.968 2.864923 secs
## 3     25     70    0.904 2.948391 secs
## 4     80     10    0.920 2.988176 secs
## 5     80     50    0.776 3.029135 secs
## 6     80     70    0.996 3.091990 secs
## 7    100     10    0.208 3.046560 secs
## 8    100     50    0.992 3.104046 secs
## 9    100     70    0.980 3.258922 secs
```

So the best neural network in terms of accuracy and CPU time is the one defined as:

```
opt.nn <- res.grid[with(res.grid,order(-Accuracy,Time)),][1,]
opt.nn
```

```
##   Layer1 Layer2 Accuracy         Time
## 6     80     70    0.996 3.09199 secs
```

And the final model of the neural network is the following one:

```
opt.mod <- keras_model_sequential()
    opt.mod %>%
        layer_dense(units = dim(x_train)[1], activation = 'relu',
                    input_shape = dim(x_train)[2]) %>%
        layer_dense(units = opt.nn$Layer1, activation = 'relu') %>%
        layer_dense(units = opt.nn$Layer2, activation = 'relu') %>%
        layer_dense(units = 5, activation = 'softmax') %>%
        compile(
                optimizer = 'rmsprop',
                loss = 'categorical_crossentropy',
                metrics = c('accuracy')
                )

start_time <- Sys.time()
opt.NN <- opt.mod %>% fit(as.matrix(x_train), y.res, epochs = 100)
end_time <- Sys.time()
time0 <- end_time - start_time
```

Then, we proceed with the first neural network using HOG descriptors. The structure is defined as follows:

- Layer 1: 81 units (input layer)
- Layer 2: 10 units (hidden layer)
- Layer 3: 5 units (output layer)

```
hog.mod.1 <- keras_model_sequential()
hog.mod.1 %>%
  layer_dense(units = 81, activation = 'relu', input_shape = c(81)) %>%
  layer_dense(units = 10, activation = 'relu') %>%
  layer_dense(units = 5, activation = 'softmax') %>%
  compile(
    optimizer = 'rmsprop',
    loss = 'categorical_crossentropy',
    metrics = c('accuracy')
  )

start_time <- Sys.time()
o.hog.1 <- hog.mod.1 %>% fit(as.matrix(o.hog), y.res, epochs = 100)
end_time <- Sys.time()
time1 <- end_time - start_time
```

Finally, we train the second neural network with HOG descriptors defined as follows:

- Layer 1: 81 units (input layer),
- Layer 2: 50 units (hidden layer),
- Layer 3: 25 units (hidden layer),
- Layer 4: 5 units (output layer).

```
hog.mod.2 <- keras_model_sequential()
hog.mod.2 %>%
  layer_dense(units = 81, activation = 'relu', input_shape = c(81)) %>%
  layer_dense(units = 50, activation = 'relu') %>%
  layer_dense(units = 25, activation = 'relu') %>%
  layer_dense(units = 5, activation = 'softmax') %>%
  compile(
    optimizer = 'rmsprop',
    loss = 'categorical_crossentropy',
    metrics = c('accuracy')
  )

start_time <- Sys.time()
o.hog.2 <- hog.mod.2 %>% fit(as.matrix(o.hog), y.res, epochs = 100)
end_time <- Sys.time()
time2 <- end_time - start_time
```

Question 2. Compare the three NN model at least in two characteristics: the performance and CPU time.

We will compare the performances of the three models considering the accuracy and the CPU time. The results obtained are the following ones:

```
performance <- data.frame(
  Model = c(paste("NN ",paste(opt.nn$Layer1,
                              opt.nn$Layer2, sep = "/")),"HOG 1","HOG 2"),
  Accuracy=c(opt.NN$metrics$acc[length(opt.NN$metrics$acc)],
             o.hog.1$metrics$acc[length(o.hog.1$metrics$acc)],
             o.hog.2$metrics$acc[length(o.hog.2$metrics$acc)]),
  Time = c(time0,time1,time2))
performance
```

```
##       Model Accuracy          Time
## 1 NN  80/70    1.000 3.126404 secs
## 2     HOG 1    0.320 2.054954 secs
## 3     HOG 2    0.356 2.288459 secs
```

It can be seen that the best model in terms of accuracy is the one obtained from the neural network with the first hidden layer of 50 and the second with 100.

On the other hand, in terms of computational time, the fastest model is the HOG 2 defined as:

- Layer 1: 81 units (input layer),
- Layer 2: 50 units (hidden layer),
- Layer 3: 25 units (hidden layer),
- Layer 4: 5 units (output layer).

But, considering the low percentage in accuracy of this model it can be discarded as the best one.

Finally, we can conclude that the best model is the one obtained from the NN 50/100 not only in terms of accuracy but also in terms of computational time.

2. Deep Learning architectures.

Categories: airplanes (800), Motorbikes (798) and Faces (435) are large categories. For each category reshape grayscale images to 32x32. Split the dataset in two halves (training and test). From the training part, fit

deep learning models according with the following architectures:

1. Convolution layer with filters = 20, kernel_size = c(5,5), activation = relu, Pooling layer with: pool_size = c(2, 2).
2. Convolution layer with filters = 40, kernel_size = c(3,3), activation = relu, Pooling layer with: pool_size = c(2, 2).
3. Fully connected layer with: units = 128, activation = relu.
4. Output layer with: units = num_classes, activation = relu.

First, we will start by importing the images into the R environment.

First we must define the path to import the desired images.

```
names <- c('airplanes', 'Motorbikes', 'Faces')
root <- "101_ObjectCategories/"
path <- c()
for (i in names){
  path <- c(path, paste(root,i, sep = ""))
}

file.path <- c()
for (i in path){
  for (files in (list.files(i))){
    file.path <- c(file.path, paste(i, files, sep = "/"))
  }
}

path.filtered <- c()
for (i in file.path){
  if (endswith(i, '.jpg', ignore.case = FALSE) == TRUE){
    path.filtered <- c(path.filtered, i)
    }
}
```

Once the paths are established, we can start importing all the images from each category and transform them into the grey scale and size 32 x 32.

```
m <-  1
output <- array(numeric(), dim = c(length(path.filtered), 32, 32, 1))

for (i in path.filtered){
  image <- readImage(i)
  if (is.na(dim(image)[3]) == FALSE){
    image.grey <- rgb_2gray(image)
    }else{
    image.grey <- image
    }

  output[m, , , ] <- resizeImage(image.grey, width = 32, height = 32, method = 'bilinear')
  m <- m + 1
}
```

Finally, we will define the three categories of the images.

```
y.res <- array(c(c(rep(0, 800)), c(rep(1, 798)), c(rep(2, 435)))) %>%
            array(dim = c(2033)) %>%
            to_categorical(num_classes = 3)
```

Once the data to work with is prepared, we will generate a training and testing sample.

```
smp_size <- floor((3/4)*2033)
index <- sample(seq_len(dim(output)[1]),  size = smp_size)

y_train <- y.res[index, ]
y_test <- y.res[-index, ]

x_train <- output[index, , , , drop = FALSE]
x_test <- output[-index, , , , drop = FALSE]
```

Now, we will build and train the proposed CNN.

```
model.cnn <- keras_model_sequential() %>%
layer_conv_2d(filters = 20, kernel_size = c(5, 5), strides = 1, padding = "valid",
              activation = "relu", input_shape = c( 32, 32, 1)) %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_conv_2d(filters = 40, kernel_size = c(3, 3), strides = 1, padding = "valid",
              activation = "relu") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_flatten() %>%
layer_dense(units = 128, activation = "relu") %>%
layer_dense(units = 3, activation = 'softmax') %>%
compile(
                optimizer = 'rmsprop',
                loss = 'categorical_crossentropy',
                metrics = c('accuracy')
                  )

output.cnn <- model.cnn %>% fit(x_train, y_train, epochs = 10)
```

We will save the model for future applications.

```
model.cnn %>% save_model_hdf5("cnn_model.h5")
```

Question 3. Assess the performance of the CNN predicting the categories of test images and obtain the confusion matrix.

First, we must generate the prediction of the model over the validation sample.

```
y.pred <- model.cnn %>% predict_classes(x_test)
```

Then, we generate the confusion matrix.

```
y.orig <- apply(y_test, 1, function(row) which(row==max(row))-1)
accuracy<-confusionMatrix(as.factor(y.orig), as.factor(y.pred))
accuracy
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2
##          0 198   1   1
##          1   4 199   0
##          2   0   0 106
##
## Overall Statistics
```

```
##
##                Accuracy : 0.9882
##                  95% CI : (0.9745, 0.9957)
##     No Information Rate : 0.3969
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9817
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity            0.9802   0.9950   0.9907
## Specificity            0.9935   0.9871   1.0000
## Pos Pred Value         0.9900   0.9803   1.0000
## Neg Pred Value         0.9871   0.9967   0.9975
## Prevalence             0.3969   0.3929   0.2102
## Detection Rate         0.3890   0.3910   0.2083
## Detection Prevalence   0.3929   0.3988   0.2083
## Balanced Accuracy      0.9868   0.9910   0.9953
```

With this model we are able to obtain an accuracy of 0.9882122 over the validation sample, this value gives us an amazing confidence for future predictions of airplanes, motorbikes or faces guaranteeing an extremly high succes rate in the classification.

Question 4. Implement an script that classifies a new image that belongs to one of the three categories: airplanes, motorbikes and faces, using the trained CNN network. The script should accept as input an image in png format, resize the image to 32 x 32 and apply the CNN to get the classification.

We will start with the function implementation.

```
classify <- function(path) {

  image <- readImage(path)

  if (is.na(dim(image)[3]) == FALSE){
    image.grey <- rgb_2gray(image)
    } else {
      image.grey <- image
      }

  image.resized <- resizeImage(image.grey, width = 32, height = 32, method = 'bilinear')
  new.image <- array(image.resized, dim = c(1, 32, 32, 1))

  cnn.model <- load_model_hdf5("cnn_model.h5")
  pred <- cnn.model %>% predict(new.image)

  if (which.max(pred) == 1) {
    print("Airplane")
    } else if (which.max(pred) == 2) {
    print("Motorbike")
    } else {
    print("Face")
```

```
    }
}
```

Now we will test the function with an image of each category:

```
#Airplane example
classify("testpics/airplane.png")
```

```
## [1] "Airplane"
```

```
#Motorbike example
classify("testpics/moto.png")
```

```
## [1] "Motorbike"
```

```
#Face example
classify("testpics/cara.png")
```

```
## [1] "Face"
```

Question 5. For all images, represent the deep features in layer 3 using PCA or t-SNE to visualize the categories representation in the plot.

The images will be converted into grey scale with size 32x32 and arranged as vectors.

```
#Import images
names <- c('airplanes', 'Motorbikes', 'Faces')
root <- "101_ObjectCategories/"
path <- c()
for (i in names){
  path <- c(path, paste(root,i, sep = ""))
}

file.path <- c()
for (i in path){
  for (files in (list.files(i))){
    file.path <- c(file.path, paste(i, files, sep = "/"))
  }
}

path.filtered <- c()
for (i in file.path){
  if (endswith(i, '.jpg', ignore.case = FALSE) == TRUE){
    path.filtered <- c(path.filtered, i)
    }
}

m <-  1
output <- matrix(0, nrow = length(file.path), ncol = 1024)

#convert to grey scale with size 32 x 32 as a vector
for (i in file.path){
  image <- readImage(i)
  if (is.na(dim(image)[3]) == FALSE){
    image.grey <- rgb_2gray(image)
    }
  else{
```

```
    image.grey <- image
    }

  img.resh <- resizeImage(image.grey, width = 32, height = 32, method = 'bilinear') #reshape
  img.vector <- as.vector(img.resh) # as 1024 vector
  output[m, ] <- img.vector
  m <- m + 1
}

#normalize the data
x_train <- as.data.frame(lapply(as.data.frame(output), normalizer))
```

Then, the class vector is generated.

```
y.res <- array(c(rep(0, 800),rep(1, 798),rep(2, 435)))
```

Once the data is prepared, we proceed with the model construction for the encoder and decoder.

```
#### Encoder
model_enc <- keras_model_sequential()
model_enc %>%
  layer_dense(units = 1024, activation = 'relu', input_shape = ncol(x_train)) %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 50, activation = 'relu')

#### Decoder
model_dec <- keras_model_sequential()
model_dec %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 1024, activation = 'relu') %>%
  layer_dense(units = ncol(x_train), activation = "relu")

#### Autoencoder
model<-keras_model_sequential()
model %>%model_enc%>%model_dec
```

```
## Model
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## sequential_13 (Sequential)       (None, 50)                    1157150
## ___________________________________________________________________________
## sequential_14 (Sequential)       (None, 1024)                  1158124
## ===========================================================================
## Total params: 2,315,274
## Trainable params: 2,315,274
## Non-trainable params: 0
## ___________________________________________________________________________
```

Once the model is defined, we train the model with the information in x_train to predict x_train.

```
model %>% compile(
  loss = "mean_squared_error",
  optimizer = "adam",
  metrics = c("mean_squared_error")
)
```

```
history <- model %>% fit(
  x= as.matrix(x_train), y = as.matrix(x_train),   # Autoencoder
  epochs = 15, batch_size = 128,
  validation_split = 0.2
)
```

When the model is trained, we proceed with the prediction of the encoder over the data to get the deep features in the layer three of the images.

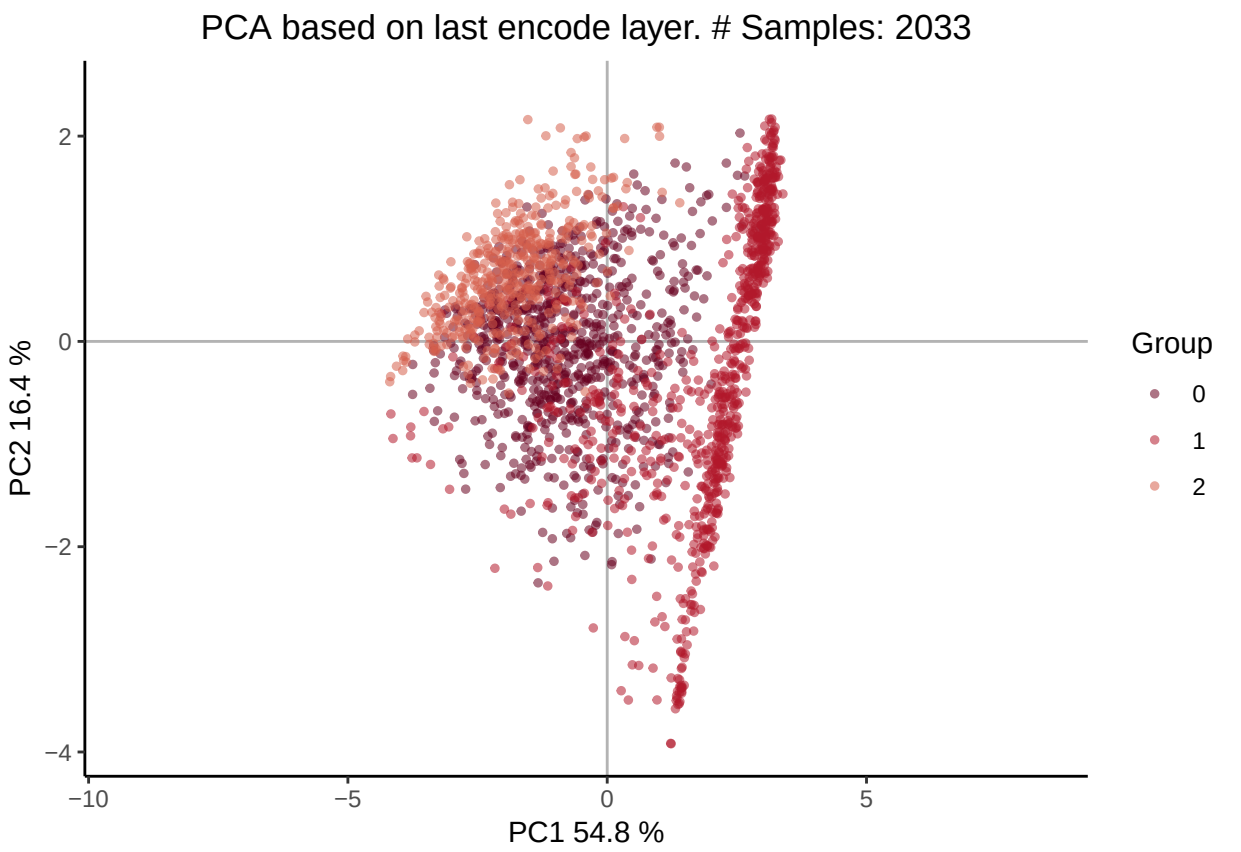
```
enc_output_cifra<-predict(model_enc,as.matrix(x_train))
dim(enc_output_cifra)
```

```
## [1] 2033   50
```

First we will execute the PCA representation.

```
plotPCA3 <- function (datos, labels, factor,title,scale,colores, size = 2, glineas = 0.25) {
  data <- prcomp(datos , scale = scale)
  dataDf <- data.frame(data$x)
  Group <- factor
  loads <- round(data$sdev^2/sum(data$sdev^2)*100,1)
  # the graphic
  p1 <- ggplot(dataDf,aes(x=PC1, y=PC2)) +
    theme_classic() +
    geom_hline(yintercept = 0, color = "gray70") +
    geom_vline(xintercept = 0, color = "gray70") +
    geom_point(aes(color = Group), alpha = 0.55, size = 1) +
    coord_cartesian(xlim = c(min(data$x[,1])-5,max(data$x[,1])+5)) +
    scale_fill_discrete(name = "")
  # the graphic with ggrepel
  p1 + geom_text_repel(aes(y = PC2 + 0.25, label = labels),segment.size = 0.25, size = size) +
    labs(x = c(paste("PC1",loads[1],"%")),y=c(paste("PC2",loads[2],"%"))) +
    ggtitle(paste("PCA based on", title, sep=" "))+
    theme(plot.title = element_text(hjust = 0.5)) +
    scale_color_manual(values=colores)
}

k <- nrow(enc_output_cifra)
scale = FALSE
plotPCA3(datos = enc_output_cifra[1:k,],
         labels = rep("",k),
         factor = as.factor(y.res[1:k]),
         scale = scale,
         title = paste ("last encode layer.", "# Samples:", k),
         colores = brewer.pal(n = 10, name = "RdBu"))
```

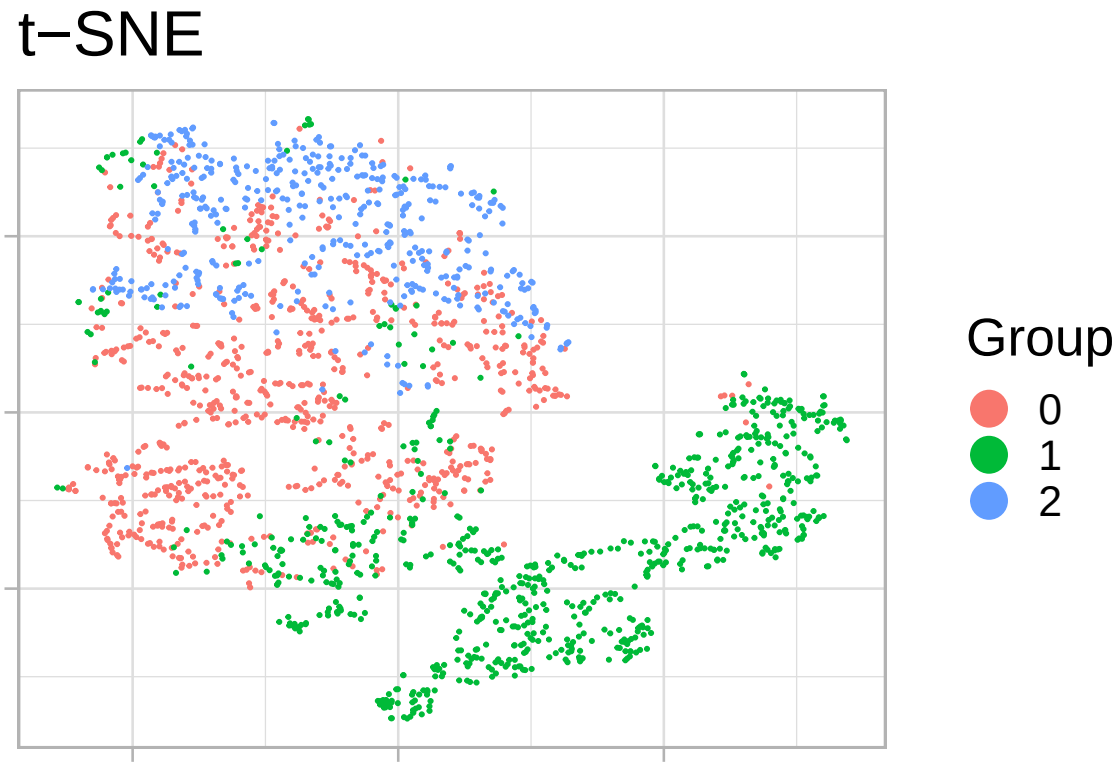


Then, the t-SNE representation.

```
tsne_model_1 = Rtsne(enc_output_cifra[1:k,],
                     check_duplicates=FALSE,
                     pca=TRUE,
                     perplexity=30,
                     theta=0.5,
                     dims=2)

## getting the two dimension matrix
d_tsne_1 = as.data.frame(tsne_model_1$Y)

## plotting the results without clustering
ggplot(d_tsne_1, aes(x=V1, y=V2, colour=as.factor(y.res[1:k]))) +
  geom_point(size=0.40) +
  guides(colour=guide_legend(override.aes=list(size=6),title="Group")) +
  xlab("") + ylab("") +
  ggtitle("t-SNE") +
  theme_light(base_size=20) +
  theme(axis.text.x=element_blank(),
        axis.text.y=element_blank())
```



If we compare the 2 representations, we can see that the results obtained from the t-SNE representation are much better in terms of discriminating the clusters than the ones obtained with the PCA.This could be because the PCA fails when a non-linear discrimination path must be found in order to separate the clusters, meanwhile the t-SNE doesn't.

It could also be because the t-SNE algorithm computes the probabilities of similarity of points in the high-dimensional space and also for the similarity points in the low-dimensional sapce. This similarity of points is calculated as the conditional probability that a point A would choose point B as its neighbor if neighbors were picked in proportion to their probability density function under a Gaussan centered at point A. Then the algorithm tries to minimize the difference between these conditional probabiliites.