OCULAR DISEASE RECOGNITION

Olgiati Alberto, matricola 814873 Raimondi Jessica, matricola 808631
Silvestrelli Sara, matricola 815664

25/4/2021

PREPROCESSING

Le immagini da caricare, presenti in un'unica cartella, sono state splittate in tre diverse cartelle di training, validation, test. Durante questa procedura le immagini appartenenti a classi sbilanciate sono state duplicate nel training set modificandone parametri come luminosità e tonalità e, in alcuni casi random, ruotandola. Ogni immagine, se raffigurante il fondo oculare di un occhio destro, è stata ruotata di 180°. Sono state escluse alcune immagini rilevate in maniera erronea.

```
rm(list = ls())
setwd("C:/Users/user/Desktop")

library(stringr)
library(magick)

set.seed(1234)
zippath <- ".\\preprocessed_images.zip"

#TRANING, VALIDATION e TEST FOLD-----------------------------------

#Crea le cartelle dove vengono archiviate le immagini per il dataset
#di traning, validation, test, classe O
train_dir <- file.path(".\\train")
dir.create(train_dir)
dir.create(".\\cp")

train_N_dir<-file.path(train_dir, "N");dir.create(train_N_dir)
train_D_dir<-file.path(train_dir, "D");dir.create(train_D_dir)
train_G_dir<-file.path(train_dir, "G");dir.create(train_G_dir)
train_C_dir<-file.path(train_dir, "C");dir.create(train_C_dir)
train_A_dir<-file.path(train_dir, "A");dir.create(train_A_dir)
train_M_dir<-file.path(train_dir, "M");dir.create(train_M_dir)
train_H_dir<-file.path(train_dir, "H");dir.create(train_H_dir)
O_dir <- file.path(".\\O") ; dir.create(O_dir)

validation_dir <- file.path(".\\validation")
dir.create(validation_dir)

valid_N_dir<-file.path(validation_dir, "N");dir.create(valid_N_dir)
valid_D_dir<-file.path(validation_dir, "D");dir.create(valid_D_dir)
valid_G_dir<-file.path(validation_dir, "G");dir.create(valid_G_dir)
```

```
valid_C_dir<-file.path(validation_dir, "C");dir.create(valid_C_dir)
valid_A_dir<-file.path(validation_dir, "A");dir.create(valid_A_dir)
valid_M_dir<-file.path(validation_dir, "M");dir.create(valid_M_dir)
valid_H_dir<-file.path(validation_dir, "H");dir.create(valid_H_dir)


test_dir <- file.path(".\\test")
dir.create(test_dir)

test_N_dir<-file.path(test_dir, "N");dir.create(test_N_dir)
test_D_dir<-file.path(test_dir, "D");dir.create(test_D_dir)
test_G_dir<-file.path(test_dir, "G");dir.create(test_G_dir)
test_C_dir<-file.path(test_dir, "C");dir.create(test_C_dir)
test_A_dir<-file.path(test_dir, "A");dir.create(test_A_dir)
test_M_dir<-file.path(test_dir, "M");dir.create(test_M_dir)
test_H_dir<-file.path(test_dir, "H");dir.create(test_H_dir)

#LABELS e SPLIT in TRAIN,VALIDATION e TEST--------------------------------
dt<-read.csv(".\\full_df.csv")
dt<-dt[,c("ID","labels","filename")]

#il dataset di traning contiene 60% dei pazienti. La divisione nei tre dataset
#avviene selezionando in modo casuale gli id dei pazienti in modo tale che
#un individuo sia presente solo in una delle tre partizioni.

ids<-dt$ID[!duplicated(dt$ID)]
l<-length(ids)
id_train<-sample(ids,l*0.6,replace=F) #trova gli id per il train
ids<-ids[which(!ids %in% id_train)]
l<-length(ids)
id_val<-sample(ids,l*0.6,replace=F) #trova gli id per il validation
ids<-ids[which(!ids %in% id_val)]
id_test<-ids  #trova gli id per il test

ID<-data.frame(ID=c(id_train,id_val,id_test),
               dataset=c(rep("train",length(id_train)),
                         rep("val",length(id_val)),
                         rep("test",length(id_test))))

write.csv(ID,".\\ID_split.csv",row.names=F)
names.files <- unzip(zippath, list=TRUE)[-1,]

#immagini eliminate: le immagini selezionate sono state scattate in modo errato.

el<-c("1086_left.jpg" , "1086_right.jpg" , "1207_left.jpg" ,
      "1207_right.jpg" , "1209_left.jpg" , "1211_right.jpg" ,
      "1212_right.jpg" , "1260_right.jpg" , "1406_right.jpg" ,
      "150_left.jpg" ,  "150_right.jpg" , "1681_left.jpg" ,
      "2173_left.jpg" , "2210_right.jpg" , "2368_left.jpg" ,
      "2368_right.jpg" , "2552_left.jpg" , "2552_right.jpg" ,
      "3330_left.jpg" , "3955_left.jpg" , "3955_right.jpg" ,
      "4176_left.jpg"  ,"4176_right.jpg" , "854_left.jpg")
```

```
#funzione per suddividere le immagini:
#input--> x: percorso dell'immagine da leggere
       #  dt: il dataset originale che contiene id, nome_foto e label
       #  augmentation_flip: vettore di lunghezza pari al numero di foto che contiene
       #                     valori generati casualmenti compresi tra 0,1.
       #                     Se il valore corrispondente ad una immagine è
       #                     maggiore di 0.5 allora l'immagine viene modificata
       #                     ruotandola orizzonatalemente e aumentandone
       #                     i parametri brightness, hue e saturation
       #  k: contatore
#output-->la funzione salva le immagini nelle cartelle in base alla label e nel caso di
         #categorie poco rappresentate salva altre due immagini modficate nei parametri
         #brightness, hue e saturation e casualmente ruotata orizzonatalmente

split_images<-function(x,dt,augmentation_flip,cont){

file_da_leggere<-unzip(zippath, x) #legge l'immagine
p<-unlist(str_split(file_da_leggere,"/"))
fname<-p[3]
id_immagine<-gsub("_right.jpg","",fname)
id_immagine<-gsub("_left.jpg","",id_immagine)
y<-gsub("[^0-9A-Za-z///' ]","",subset(dt,filename==fname)$labels)
y<-eval(parse(text = y)) #identifico la label

#carico l'immagine
img<-image_read(file_da_leggere)

#se l'immagine è destra viene ruotata di 180 gradi
if ("right.jpg" %in% unlist(str_split(fname,"_")) ){
  img<-image_flop(img)
}

if(!fname %in% el) {

  if(y=="0") { #escludo le immagini con label 0
    image_write(img,paste(0_dir,"//",fname,sep=""))
  }
  else{

      #salva l'immagine nel train
      if (id_immagine %in% id_train){
      train_dir_by_class<-paste(train_dir,"//",y,sep="")
      image_write(img,paste(train_dir_by_class,"//",fname,sep=""))

      #Se la label è sbilanciata inserisci 2 foto in più modificate
        if (y=="A" | y=="C" | y=="G" | y=="M" | y=="H"){
        img1<-img
        #impostazioni di modifica della foto
        brightness<-110
        hue=95
        saturation=105
        if (augmentation_flip[cont]>0.5) {
           img1<-image_flip(img) #ruota orizzontalmente
```

```
            hue=105
            saturation=95
        }
        #salva le immagini modificate
        img1<-image_modulate(img1,
                             brightness = brightness,
                             saturation=saturation,
                             hue=hue)
        image_write(img1,paste(train_dir_by_class,"//","aug1_",fname,sep=""))
        img2<-image_modulate(img1, brightness = brightness-5,
                             saturation=saturation-2, hue=hue+3)
        image_write(img2,paste(train_dir_by_class,"//","aug2_",fname,sep=""))
        }
      }
      #salva l'immagine nel validation
      if (id_immagine %in% id_val){
        valid_dir_by_class<-paste(validation_dir,"//",y,sep="")
        image_write(img,paste(valid_dir_by_class,"//",fname,sep=""))
      }
      #salva l'immagine nel test
      if (id_immagine %in% id_test){
        test_dir_by_class<-paste(test_dir,"//",y,sep="")
        image_write(img,paste(test_dir_by_class,"//",fname,sep=""))
      }

      }
    }
}

nobs<-nrow(names.files)
phi<-runif(nobs) #vettore per definire la rotazione della foto sopra/sotto


#ciclo for per leggere tutte le foto
for (k in 1:nobs) {
  split_images(names.files$Name[k],dt,phi,k)
}
```

PRE TRAINED NEURAL NETWORK

Definiamo la rete preallenata escludendo il top layer.

```
library(keras)
#install_keras()
library(tensorflow)

#per rendere il codice riproducibile

seed=1234
set.seed(seed)
reticulate::py_set_seed(seed)
tf$random$set_seed(seed)
```

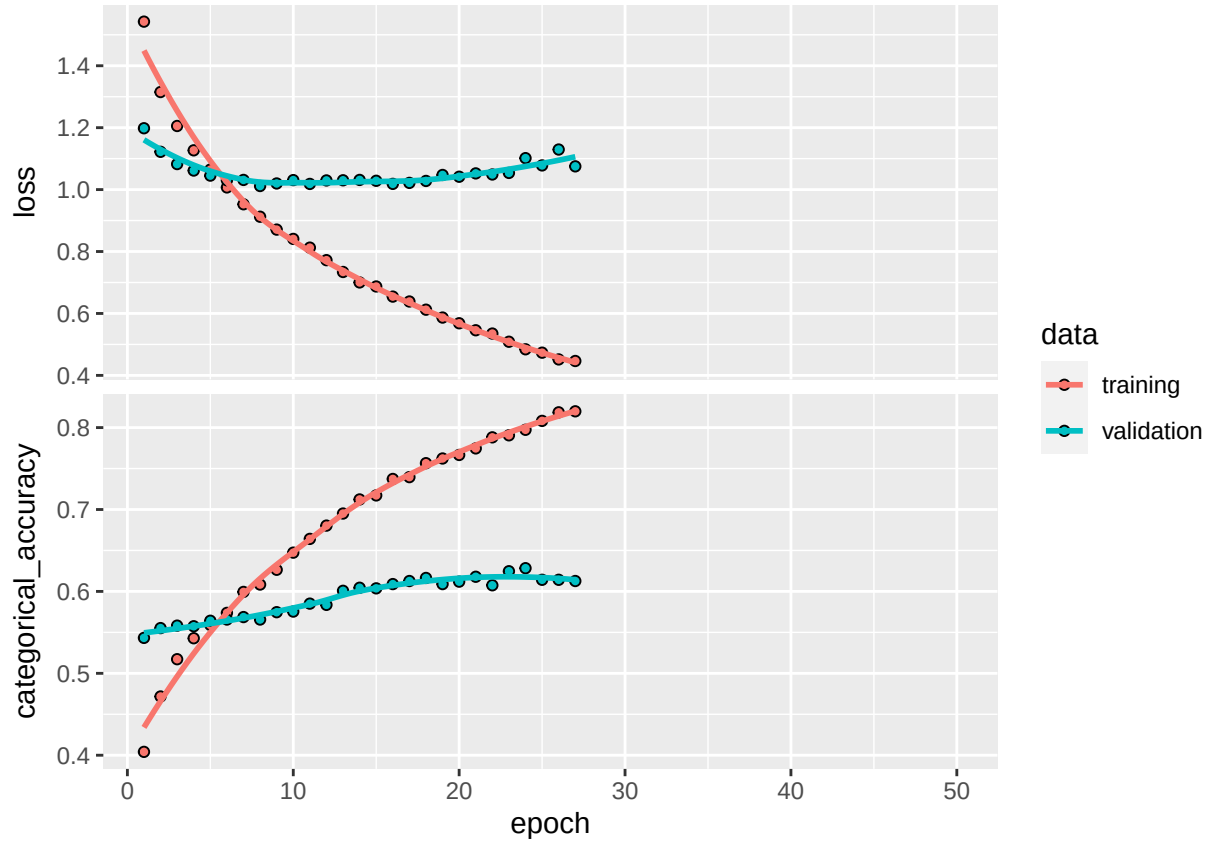
```r
#size delle immagini

x<- 224
y<- 224

#batch size
b<-20

# Le immagini vengono di seguito riscalate (1/255)
train_datagen <- image_data_generator(rescale = 1/255)
validation_datagen <- image_data_generator(rescale = 1/255)
test_datagen <- image_data_generator(rescale =1/255)


# definiamo i percorsi

train_dir <- ".\\train"

validation_dir <- ".\\validation"

test_dir <- ".\\test"


# rete pre allenata mobilenet v2

conv_base <- application_mobilenet_v2(
  # Pesi da utilizzare per iniziallizzare la rete
  weights = "imagenet",
  # Questo comando serve ad includere o meno il layere fully connected
  include_top = FALSE,
  # Dimensione del tensore che utilizzeremo come input della nostra rete
  input_shape = c(x, y, 3)
)

extract_features <- function(directory, sample_count,batch_size) {
  datagen <- image_data_generator(rescale = 1/255)
  features <- array(0, dim = c(sample_count, 7, 7, 1280))
  labels <- array(0, dim = c(sample_count,7))

  generator <- flow_images_from_directory(
    directory = directory,
    generator = datagen,
    target_size = c(x, y),
    batch_size = batch_size,
    class_mode = "categorical"
  )

  i <- 0
  while(TRUE) {
    batch <- generator_next(generator)
    inputs_batch <- batch[[1]]
    labels_batch <- batch[[2]]
    features_batch <- conv_base %>% predict(inputs_batch)
```

```
    index_range <- ((i * batch_size)+1):((i + 1) * batch_size)
    features[index_range,,,] <- features_batch
    labels[index_range,] <- labels_batch
    i <- i + 1
    # Per bloccare il ciclo
    if (i * batch_size >= sample_count)
      break
  }

  list(
    features = features,
    labels = labels
  )
}

#4892 immagini in totale nel training
train <- extract_features(train_dir,4880,batch_size=b)
#1349 immagini in totale nel validation
validation <- extract_features(validation_dir, 1340,batch_size=b)
#912 immagini in totale nel test
test <- extract_features(test_dir, 900,batch_size=b)

reshape_features <- function(features) {
  array_reshape(features, dim = c(nrow(features),7 * 7 * 1280))
}

train$features <- reshape_features(train$features)
validation$features <- reshape_features(validation$features)
test$features <- reshape_features(test$features)


#salviamo l'environment
save(test,train,validation, file = "MobileNetV2feauture.RData")
```

RETE FULLY CONNECTED

```
set_random_seed(123)

model <- keras_model_sequential() %>%

  layer_dense(units = 224, activation = "relu",
              input_shape = 7 * 7 * 1280) %>%
  layer_dense(units = 160, activation = "relu") %>%
  layer_dropout(rate = 0.45) %>%
  layer_dense(units = 48, activation = "relu") %>%
  layer_dense(units = 7, activation = "sigmoid")

model %>% compile(
  optimizer = optimizer_rmsprop(lr = 1e-5),
  loss = "categorical_crossentropy",
  metrics = "categorical_accuracy"
```

```
)

checkpoint_path<-".\\cp\\modello_vincente.hdf5"

# Create checkpoint callback
cp_callback <- callback_model_checkpoint(
  filepath = checkpoint_path,
  monitor = "val_categorical_accuracy",
  mode='max',
  save_weights_only = FALSE,
  save_best_only = TRUE,
  verbose = 0
)

early_stopping<-callback_early_stopping(
                  monitor = "val_categorical_accuracy",
                  patience = 3,
                  verbose = 0,
                  mode = "max",
                  restore_best_weights = FALSE)

history <- model %>% fit(
  train$features, train$labels,
  epochs = 50,
  batch_size = 100,
  validation_data = list(validation$features, validation$labels),
  callbacks = list(cp_callback, early_stopping)
)

plot(history)
```




```
#numero di epoche considerate
(early_stop<-length(history$metrics$val_loss))
```

```
[1] 27
```

```
#salva il modello che massimizza accuracy
#nel dataset di validation
new_model <- load_model_tf(checkpoint_path)
previsione<-new_model %>% predict(test$features)

classi<-c("AMD","Cataract","Diabetes","Glaucoma","Hypertension","Myopia","Normal")
classe_prevista<-apply(previsione,1,which.max)
classe_osservata<-apply(test$labels,1,which.max)


# funzione che trasforma un numero intero nella rispettiva classe
int_to_class<-function(x,classi){
  y<-c()
  for (i in 1:length(x)) {
    cl_i<-classi[x[i]]
    y<-c(y,cl_i)
  }
  y<-as.factor(y)
  return(y)
}
```

```r
classe_prevista<-int_to_class(classe_prevista,classi)
classe_osservata<-int_to_class(classe_osservata,classi)

require(caret)
c0<-confusionMatrix(classe_prevista,classe_osservata)
round(c0$byClass,4)[,c(1,2,11)] #statistiche byClass
```

```
                     Sensitivity Specificity Balanced Accuracy
Class: AMD                0.3929      0.9874            0.6901
Class: Cataract           0.8293      0.9837            0.9065
Class: Diabetes           0.2941      0.9395            0.6168
Class: Glaucoma           0.2174      0.9895            0.6034
Class: Hypertension       0.0526      0.9955            0.5240
Class: Myopia             0.8409      0.9907            0.9158
Class: Normal             0.8951      0.4711            0.6831
```

```r
c0$table #confusion matrix
```

```
             Reference
Prediction     AMD Cataract Diabetes Glaucoma Hypertension Myopia Normal
  AMD           11        0        7        0            2      0      2
  Cataract       0       34        1        1            0      2     10
  Diabetes       1        2       75        3            6      0     27
  Glaucoma       1        1        2       10            0      0      5
  Hypertension   0        0        3        0            1      0      1
  Myopia         0        0        4        0            0     37      4
  Normal        15        4      163       32           10      5    418
```

```r
round(c0$overall,4)[1:2] #Accuracy e Kappa sul test
```

```
Accuracy    Kappa
  0.6511   0.4029
```