SISTEMAS INTELIGENTES PARA LA GESTIÓN EN LA EMPRESA

MASTER PROFESIONAL EN INGENIERÍA EN INFORMÁTICA

Práctica 2: Multiclasificación

Autores
José Ángel Díaz García y Pablo Martin-Moreno

Equipo
José Ángel & Pablo

Puntuación y Posición
0.82297

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Granada, Junio de 2017

Índice general

1. Introducción **5**

1.1. Problema y Dataset 5

1.1.1. Evaluación 6

1.2. Herramientas y objetivos 6

1.2.1. Hardware 6

1.2.2. Software 7

1.2.3. Objetivos 7

1.3. Organización del trabajo 8

2. Preprocesado **9**

2.1. Resize data 9

2.2. Cut data 11

2.3. Data Aumentation 13

2.4. Undersampling 15

3. Clasificación con NN **17**

3.1. Intel Deeplearning SDK 17

3.2. From Scratch 18

3.2.1. Mxnet 18

3.2.2. TensorFlow+Keras 27

3.3. Red ya entrenada 30

3.4. Fine Tunning 33

4. Multiclasificación y mapas de características 34

4.1. Multiclasificacion 34

4.2. Mapas de características 34

5. Conclusiones y vías futuras 35

5.1. Conclusiones finales 35

5.2. Vías futuras 35

6. ANEXO I 36

Índice de figuras

2.1. Deformación de la imagen al redimensionar. 11
2.2. Cortado de la imagen. 11

3.1. Type 1 imagen 0, clasificada como Weimaraner (perro). 31
3.2. Type 1 imagen 1168, clasificada como Pomegranate (granada). 32
3.3. Type 2 imagen 64, clasificada como Pomegranate (granada). . 32
3.4. Type 2 imagen 913, clasificada como bubble, (burbuja). 32
3.5. Type 3 imagen 1127, clasificada como Pomegranate (granada). 33
3.6. Type 3 imagen 71, clasificada como Meat Shop (carnicería). . 33

Índice de tablas

1.1. Especificaciones técnicas de la máquina 1. 6
1.2. Especificaciones técnicas de la máquina 2. 7

6.1. Tabla de resultados 37

Capítulo 1

Introducción

Esta última práctica está enmarcada dentro de la asignatura **Sistemas Inteligentes para La Gestión en la Empresa** del Master Profesional en Ingeniería Informática de la UGR y aborda un problema real de predicción multiclase en la plataforma Kaggle [2].

Este problema, es de un nivel avanzado, y a lo largo de los siguientes capítulos intentaremos aportar una solución aceptable en la plataforma Kaggle, así como estudiar y asentar los diferentes conceptos teóricos vistos en la asignatura.

1.1. Problema y Dataset

El problema en última instancia es un problema de clasificación multiclase real el cual deberá ser resuelto mediante técnicas de *deeplearning*. El problema, en concreto es Intel & MobileODT Cervical Cancer Screening [3] y trata de clasificar partiendo de imágenes del cervix de distintas pacientes, que tipo de tratamiento para el cancer es más efectivo, aspecto muy relevante sobre todo en puntos del mundo rural donde el acceso a grandes infraestructuras médicas puede estar limitado y donde la prevención en etapas tempranas puede ser decisiva.

El dataset está compuesto de la siguiente manera:

- **test**: 512 Imágenes que deberemos clasificar tras el entrenamiento.

- **train**: Tenemos un total de 1581 muestras para entrenar, compuestas por 350 de tipo 1, 781 de tipo 2 y 450 de tipo 3.
- **train-extra**: El dataset ofrece también una gran conjunto de imagenes extra para el entrenamiento de unos 30GB de espacio en disco.

Podemos ver como el dataset muestra cierto ratio de des balanceo, por lo que en instancias superiores de la práctica, deberemos atacar este punto para obtener mejores resultados. Por otro lado, las restricciones del problema hacen que sean interesantes propuestas como *One Vs One* o *One vs All* que analizáremos en sucesivos puntos.

1.1.1. Evaluación

La evaluación de está práctica tendrá como evaluador del modelo la función *logloss*, donde los falsos negativos tendrán una gran penalización. En caso de obtener una evaluación perfecta, el *logloss* del clasificador sería **0**.

1.2. Herramientas y objetivos

En esta sección veremos una breve introducción a las herramientas usadas para el desarrollo de la práctica así como de los principales objetivos que se buscan conseguir con el desarrollo de la misma.

1.2.1. Hardware

Elemento	Características
Procesador	2,6 GHz Intel Core i5
GPU	-
Memoria Ram	8 GB 1600 MHz DDR3
Disco duro	SATA SSD de 120 GB

Tabla 1.1: Especificaciones técnicas de la máquina 1.

Elemento	Características
Procesador	Intel Core i7 6700HQ
GPU	2,6 GHz Intel Core i5
Memoria Ram	16 GB SDRAM
Disco duro	128GB SSD

Tabla 1.2: Especificaciones técnicas de la máquina 2.

1.2.2. Software

El software utilizado es en su práctica totalidad software libre, siendo el restante software propietario cuyas licencias vienen incluidas en el sistema operativo de la máquina 1.1 siendo este OS X "Sierra", o el Windows 10 de la máquina 1.2. El software usado es:

- **RStudio**: Entorno de trabajo para R.
- **Tensorflow**: Entorno de deeplearning sobre Python.
- **Keras**: Capa de abstracción sobre Tensorflow.
- **MXNet**: Libreria de Deeplearning sobre R.
- **Atom**: Editor de texto plano para la programación de los scripts.
- **TeXShop**: procesador de textos basado en *Latex* usado para elaborar la documentación del presente proyecto.

1.2.3. Objetivos

Los objetivos de este trabajo podrían resumirse en los siguientes:

- Obtener un modelo predictivo fiable que dado una nueva imagen pueda predecir el tipo de tratamiento contra el cancer a aplicar.
- Obtener un valor de *LogLoss* aceptable para escalar posiciones en la competición de Kaggle.

- Comprender y estudiar las distintas técnicas de minería de datos vistas en la asignatura.
- Ahondar en el proceso de la multiclasificación y las vertientes de estudio dentro de la misma.
- Estudiar distintos métodos de clasificación sobre el mapa de características proveniente del entrenamiento de las redes neuronales.

1.3. Organización del trabajo

La organización del presente documento, se centra en detallar cada uno de los pasos seguidos durante el estudio y resolución del problema planteado en esta introducción. En el capítulo 2 veremos los scripts y explicaciones asociadas al preprocesado de datos, más concretamente al data augmentation y cierto enfoque de *imbalance learning*. En el capitulo 3 tendremos el grueso del trabajo en el que entraremos en detalle en una primera aproximación *from scracth* con *mxnet* y el grueso del trabajo realizando *fine tunning* en Tensorflow. Finalizaremos con el capítulo 4 dedicado al estudio de multiclasificación y uso de otros clasificadores y por último las conclusiones y vías futuras que quedan relegadas al capítulo 5.

Los resultados obtenidos en la competición de kaggle pueden encontrarse en el Anexo de la sección 6.

Capítulo 2

Preprocesado

En este capítulo enunciaremos el proceso de preprocesado llevado a cabo durante la realización de la práctica. Cabe destacar, que por facilitar la comprensión todo el proceso se enuncia en este capítulo, pero hay ciertos puntos como el del proceso de undersampling (sección 2.4) y el de corte de las imágenes (sección 2.2) que surgen fruto de cambios e ideas en el proceso de aprendizaje y entrenamiento de las redes neuronales.

2.1. Resize data

Como veremos en el capítulo de conclusión, uno de los requisitos y mayores problemas que esta práctica a ofrecido han sido las limitaciones de memoria tanto de principal para computo como de memoria secundaria para almacenar todas las imágenes y sus diferentes versiones. Con el fin de poder manejar estas más eficientemente se ha realizado un resize de las mismas a tamaño de 256x256px, para ello hemos seguido el siguiente script en R.

```
#------------------------------------------------------------------

# Clear workspace
rm(list=ls())

#------------------------------------------------------------------
# Load and pre-process images
#------------------------------------------------------------------
```

```
9
10 # Set run parameters parameters
11 test_img_path <- "."
12 width  <- 256
13 height <- 256
14
15
16 #------------------------------------------------------------------
17 # Load and pre-process train images
18 #------------------------------------------------------------------
19
20 # Load EBImage library
21 library(EBImage)
22
23 # Load images into a dataframe
24 library(gsubfn)
25 img_file_list <- list.files(path = test_img_path, pattern = "*.jpg", full.names =
       TRUE, recursive = TRUE)
26
27 train_df <- data.frame()
28
29 for(i in 1:length(img_file_list)) {
30   img_file_name <- img_file_list[i]
31   #img_class <- strapplyc(img_file_list[i], ".*/Type_(.*)/")[[1]]
32   img <- readImage(img_file_name)
33   img_resized <- resize(img, w=width, h=height)
34
35   name <- paste("reducidas",img_file_name, sep="/")
36   dir.create("reducidas", showWarnings = FALSE)
37   writeImage(img_resized, name)
38
39 }
```

Con este simple script podemos preprocesar las imágenes originales y reducirías de tamaño de manera que son mas fácilmente manejables, pero esto nos da un problema asociado y es que estamos modificando los datos originales.

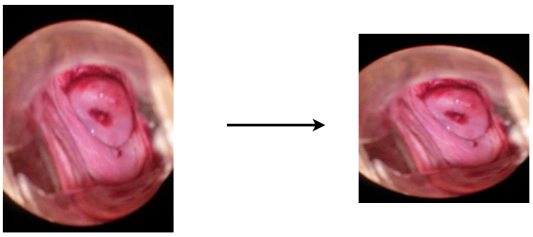

Figura 2.1: Deformación de la imagen al redimensionar.

Tal y como podemos apreciar en la figura 2.1 si una imagen rectangular la 'obligamos' a ser cuadrada, estamos deformando el contenido por lo que ya tendremos otra imagen distinta de la original, a este problema la estudiaremos en la siguiente sección.

2.2. Cut data

Siguiendo con el ejemplo de la sección anterior, la solución a la deformación de la imagen pasa por cortar al ratio de aspecto 1:1 y posteriormente redimensionar para tener imágenes de menos peso. Este paso, puede explicarse con la imagen 2.2.

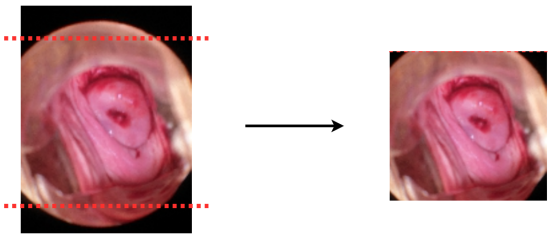

Figura 2.2: Cortado de la imagen.

La altura de la imagen debe ser por tanto igual al ancho de la misma y para ello, debemos quitar los espacios que vemos en la imagen anterior, tras lo cual tendremos una imagen cuadrada pero sin deformar el contenido, además de quitar partes de la imagen que en más del 90 % de los casos no tienen importancia. Para calcular el corte hemos usado la ecuación 2.1, que

nos da el valor que deberemos substraer de la imagen original tanto por la parte superior como por la inferior.

$$corte = \frac{height - widht}{2} \tag{2.1}$$

El script que usamos para este objetivo es el siguiente:

```
#-------------------------------------------------------------------------
# This script cut the images to prevent original data deformation
#-------------------------------------------------------------------------


#setwd("D:/Facultad/Master/Segundo cuatrimestre/SIGE/PracticaFinal")
# Clear workspace
rm(list=ls())

# Set run parameters parameters
test_img_path <- "."

# Load EBImage library
library(EBImage)
# Load images into a dataframe
library(gsubfn)
img_file_list <- list.files(path = test_img_path, pattern = "*.jpg", full.names =
    TRUE, recursive = FALSE)

img_file_list
dir.create("crop", showWarnings = FALSE)
for(i in 1:length(img_file_list)) {

  img_file_name <- img_file_list[i]
  name <- paste("crop",img_file_name, sep="/")
  print(name)
  if(!file.exists(name)){
    img <- readImage(img_file_name)
    width <- dim(img)[1]
    heigth <- dim(img)[2]
    if (width < heigth){
      cut <- (heigth-width)/2
      p <- heigth-cut
      img2 <- img[0:width,cut:p,]
    }else{
      cut <- (width-heigth)/2
      p <- width-cut
```

```
37          img2 <- img[cut:p,0:heigth,]
38        }
39
40        writeImage(img2, name)
41      }
42
43  }
44  print("Finalizado")
```

2.3. Data Aumentation

Uno de los principales problemas en *deep learning* es la falta de datos. Para ello, pueden usarse técnicas de *data aumentation* que consisten en aplicar ligeras transformaciones a las imágenes para conseguir un conjunto de entrenamiento mayor. Para ello, hemos usado dos vertientes, una con R (siguiente script) con la que obtenemos imágenes modificadas en ficheros separados para poder ir combinándolas como deseemos, y por último, en el código Python que veremos en puntos siguientes, sobre estas volvíamos a aplicar una serie de cambios aumentando por tanto el data set bastante.

```
1
2  #------------------------------------------------------------------
3
4  # Clear workspace
5  rm(list=ls())
6
7  #------------------------------------------------------------------
8  # Load and pre-process images
9  #------------------------------------------------------------------
10
11 # Set run parameters parameters
12 train_img_1_path <- "./Type_1"
13 train_img_2_path <- "./Type_2"
14 train_img_3_path <- "./Type_3"
15
16
17 #------------------------------------------------------------------
18 # Load and pre-process train images
19 #------------------------------------------------------------------
20
21 # Load EBImage library
```

```
22 library(EBImage)
23
24 # Load images into a dataframe
25 library(gsubfn)
26 img_file_1_list <- list.files(path = train_img_1_path, pattern = "*.jpg",
       full.names = TRUE, recursive = TRUE)
27 img_file_2_list <- list.files(path = train_img_2_path, pattern = "*.jpg",
       full.names = TRUE, recursive = TRUE)
28 img_file_3_list <- list.files(path = train_img_3_path, pattern = "*.jpg",
       full.names = TRUE, recursive = TRUE)
29
30
31 #train_df <- data.frame()
32
33 #Hacemos 3 bucles porque tienen tama?os distintos cada clase.
34
35 Imagenes<-function(image,type) {
36   img_file_name <- image
37
38   img <- readImage(img_file_name)
39
40   if(type=="flip"){
41     img <- flip(img)
42   }
43
44   if(type=="flop"){
45     img <- flop(img)
46   }
47
48   if(type=="rot"){
49     img = rotate(img, 90, bg.col = "white")
50   }
51   if(type=="trans"){
52     img_t = transpose(img)
53   }
54
55   #print("IMG_FILE_NAME")
56   #print(img_file_name)
57
58   salida <- substr(img_file_name, 0, 8)
59   name <- paste(salida,type, sep="/")
60   salida <- substr(img_file_name, 10,nchar(img_file_name) )
61   #print("salida")
62   #print(salida)
63   salida <- paste(type,salida, sep="-")
```

```
64
65    name2 <- paste(name,salida, sep="/")
66    #print("NAME2")
67    #print(name2)
68
69    dir.create(name, showWarnings = FALSE)
70    writeImage(img, name2)
71  }
72
73
74
75  tipos<-function(imagen){
76    Imagenes(imagen,"flip")
77    Imagenes(imagen,"flop")
78    Imagenes(imagen,"rot")
79    Imagenes(imagen,"trans")
80  }
81
82  for(i in 1:length(img_file_1_list)) {
83    tipos(img_file_1_list[i])
84  }
85
86
87  for(i in 1:length(img_file_2_list)) {
88    tipos(img_file_2_list[i])
89  }
90
91  for(i in 1:length(img_file_3_list)) {
92
93    tipos(img_file_3_list[i])
94
95  }
96
97  print("FINALIZADO")
```

2.4. Undersampling

Por último, en nuestro proceso de pre-procesado, descubrimos que las clases mostraban cierto desequilibrio. Concretamente, la clase1 frente a clase 3 de 1,66, y de clase 1 frente a clase 2 de 2,35. Aunque no es mucho, están por

encima del 1,5, valor a partir del cual se considera un problema de clase desequilibradas.

Para intentar solventar este problema, quitamos algunas imágenes de las clases mayoritarias intentando siempre que las imágenes originales estuvieran presentes entre las seleccionadas, de manera que se suprimieron algunas de las transformaciones obteniendo ratios de balanceo por debajo del 1.5. Esta solución, al evaluar en test, lejos de mejorar empeoraba el modelo ya que como hemos visto en 2.3 el principal problema a solventar en esta competición es la falta de información en training y con esta técnica, aunque igualamos clases, perdemos información que puede ser decisiva.

Capítulo 3

Clasificación con NN

En este capítulo veremos el proceso seguido y las distintas vertientes de entrenamiento usadas a lo largo de la realización de la práctica. Concretamente veremos entrenamiento *from scratch* con dos tecnologías distintas y por último *fine tuning*.

3.1. Intel Deeplearning SDK

Como el software se ve en la práctica y en virtud de probar todos y cada uno de los elementos estudiados, instalamos el software de Intel en la máquina 1.1. El primero de los problemas vino con el tiempo de instalación y la cantidad de memoria necesaria por este, por un lado para desplegar los 4 contenedores docker que montan la app, como para alojar los dataset, ya que crea una copia dentro de los contenedores de los mismos. Por otro lado, una vez instalado, constatamos la imposibilidad de validar el conjunto de test de la competición sin realizar modificaciones en los scripts de Caffe o Tensorflow o creando clases ficticias para que clasifique con las cuales el proceso de entrenamiento ofrecería resultados muy malos y la interfaz gráfica del mismo serviría mas bien de poco. La cantidad de problemas y la perdida de tiempo con el software, hizo que migráramos directamente hacia otras tecnologías como las que veremos a continuación.

3.2. From Scratch

La estrategia seguida en una competición en la que el tiempo de computo era un factor claramente privativo, ha pasado por afinar el proceso de preprocesado de datos con scripts que ofrecían resultados 'aceptables' en poco tiempo de computo. Por ello, nos hemos basado en redes neuronales muy sencillas, sobre las cuales podríamos ir probando distintas combinaciones de preprocesado, como las vistas en el capítulo 2 y ver cuales son las que mejor se adaptan a nuestro problema, con la premisa de si funcionan bien en una red sencilla también lo harán en redes más complejas. Las redes elaboradas provienen de kernels de Kaggle, concretamente la de MXNET proviene del siguiente tutorial [4] y la de TensorFlow de este otro [5], aunque si bien el código ha sido modificado para adaptarlo a nuestros requisitos.

3.2.1. Mxnet

El siguiente script ha sido el empleado para la realización del proceso from scratch con la máquina 1.1 ya que esta no dispone de GPU. De este script visto en Kaggle, obtuvimos la idea de comprobar el sesgo de las clases y el balanceo ya que en este se comprueba dicho punto.

En este script, se añaden un valor por cada uno de los colores (RGB) para cada uno de los pixeles de la imagen y se entrena con ello. La red es muy sencilla, y está lejos del estado del arte con redes convolutivas más complejas por ello, los resultados obtenidos por la misma, son del orden de 0.3XX de probabilidad a cada una de las clases, lo que implica valores de log loss muy malos y Accuracy de 0.4 lo que implica una clasificación casi aleatoria al tener un problema con tres clases.

```
1  #------------------------------------------------------------------------
2  # Kaggle Intel Cervix Cancer Challenge
3  #
4  #
5  # Image loading and basic pre-processing with EBImage
6  # Color images
7  # Submission to Kaggle is generated
8  #------------------------------------------------------------------------
9
10
11 #------------------------------------------------------------------------
```

```
12 #Load librarys
13 #------------------------------------------------------------------
14
15
16 library(dplyr)
17 library(EBImage)
18 library(mxnet)
19 library(nnet)
20
21
22 #------------------------------------------------------------------
23 # Load images using EBImage
24 #
25 # This loop resize  all  images, we dont have to do data-augmentation
26 # because, it  is  done with the script  data-augmentation.R.
27 #------------------------------------------------------------------
28
29
30 paths <- c("/Users/joseadiazg/Desktop/Temporales
       UGR/SIGE/all_data_resized/Type_1",
31             "/Users/joseadiazg/Desktop/Temporales
                  UGR/SIGE/all_data_resized/Type_2",
32             "/Users/joseadiazg/Desktop/Temporales
                  UGR/SIGE/all_data_resized/Type_3")
33
34
35 # Uncomment this if you want to load all training + extra images
36
37
38 #------------------------------------------------------------------
39 #paths <- c("/Users/joseadiazg/Desktop/Temporales
       UGR/SIGE/all_data_resized/Type_1",
40 #            "/Users/joseadiazg/Desktop/Temporales
       UGR/SIGE/all_data_resized/Type_2",
41 #            "/Users/joseadiazg/Desktop/Temporales
       UGR/SIGE/all_data_resized/Type_3",
42 #             "/Users/joseadiazg/Desktop/Temporales
       UGR/SIGE/train-extra-unidas/Type_1",
43 #             "/Users/joseadiazg/Desktop/Temporales
       UGR/SIGE/train-extra-unidas/Type_2",
44 #             "/Users/joseadiazg/Desktop/Temporales
       UGR/SIGE/train-extra-unidas/Type_3",
45 #             )
46 #------------------------------------------------------------------
47
```

```
48 in_type_counter <- 1
49
50 for (t in paths){
51
52   patients <- dir(t)
53
54   #For this simple example We rescale photos to 128*128 (= 16384)
55
56
57   n_columnas <- 1 + 1 + 49152
58
59
60   ordered_images <- data.frame(matrix(nrow = length(patients), ncol =
         n_columnas))
61   colnames(ordered_images) <- c("paciente", "Type", paste("R", (1:16384), sep =
         ""), paste("G", (1:16384), sep = ""), paste("B", (1:16384), sep = ""))
62
63   contador <- 1
64
65   if(in_type_counter == 1){
66     ordered_images$Type <- 1 #(*) ojo , asignación manual segun el folder que
           esté procesando
67   }
68
69   if(in_type_counter == 2){
70     ordered_images$Type <- 2 #(*) ojo , asignación manual segun el folder que
           esté procesando
71   }
72
73   if(in_type_counter == 3){
74     ordered_images$Type <- 3 #(*) ojo , asignación manual segun el folder que
           esté procesando
75   }
76
77   #+++
78   mom_inicio <- Sys.time()
79   print("beginning calculation: ")
80   print(mom_inicio)
81   #+++
82
83   for (p in patients){
84
85     cat("contador:", contador, " paciente:", p, "\n")
86
87     ordered_images$paciente[contador] <- p
```

```
88
89      imagen_paciente <- readImage(paste(t, p, sep = "/")) #abre imagen de cada
            paciente...
90
91      #TODO: We have to change this resize to a new version in witch we use the same
92      # proportion of tam but smaller.
93
94      imagen_paciente <- resize(imagen_paciente, w = 128, h = 128)
95
96
97      ordered_images[contador, c(3:16386)] <- imagen_paciente[, , 1]
98      ordered_images[contador, c(16387:32770)] <- imagen_paciente[, , 2]
99      ordered_images[contador, c(32771:49154)] <- imagen_paciente[, , 3]
100
101     contador <- contador + 1
102
103   }
104
105
106   #+++
107   print("calculation time: ")
108   print(Sys.time() - mom_inicio)
109   #+++
110
111
112   if(in_type_counter == 1){
113     ordered_images_Type_1 <- ordered_images
114   }
115
116   if(in_type_counter == 2){
117     ordered_images_Type_2 <- ordered_images
118   }
119
120   if(in_type_counter == 3){
121     ordered_images_Type_3 <- ordered_images
122   }
123
124
125   in_type_counter <- in_type_counter + 1
126 }
127
128
129 ordered_images_all <- bind_rows(ordered_images_Type_1,
        ordered_images_Type_2, ordered_images_Type_3)
130
```

```
131
132 #-----------------------------------------------------------------
133 # Subset for validation
134 #-----------------------------------------------------------------
135
136 table(ordered_images_all$Type)
137
138
139 ordered_images_all$Type <- ordered_images_all$Type - 1
140
141
142 set.seed(9)
143 set_validacion <- ordered_images_all %>%
144   group_by(Type) %>%
145   sample_n(25) %>%
146   ungroup()
147
148
149 #-----------------------------------------------------------------
150 # Undersampling
151 #-----------------------------------------------------------------
152
153
154 set_train_unbalanced <- ordered_images_all[ordered_images_all$paciente %in%
      setdiff(ordered_images_all$paciente, set_validacion$paciente), ]
155
156 #primera muestra train undersampled:
157 set.seed(9)
158 undersample_1_set_train <- set_train_unbalanced %>%
159   group_by(Type) %>%
160   #sample_n(225) %>%
161   ungroup() %>%
162   sample_frac(1) %>%
163   sample_frac(1) #WE SHUFFLE TWICE
164
165 # dim(undersample_1_set_train)
166 # #[1]   675 12290
167 # dim(set_validacion)
168 # #[1]    75 12288
169
170
171 #_____train and validación labels_____________________________
172 target_undersample_1 <- undersample_1_set_train$Type
173 label_set_validacion <- set_validacion$Type
174 #_____________________________________________________________
```

```
175
176
177 #_____train set1(undersampled) y and validation set, both balanced_______
178 undersample_1_set_train <-undersample_1_set_train[, c(3:49154)]
179 set_validacion <-set_validacion[, c(3:49154)]
180 #_____target y label de
        validación_____________________________________________
181
182 undersample_1_set_train <- t(undersample_1_set_train)
183 dim(undersample_1_set_train) <- c(128, 128, 3, 675)
184
185
186 set_validacion <- t(set_validacion)
187 dim(set_validacion) <- c(128, 128, 3, 75)
188
189
190
191
192 #-------------------------------------------------------------------
193 # Tunning parametters for the nn
194 #-------------------------------------------------------------------
195
196
197 n_output <- 3
198 num_filters_conv2 = 14
199 num.round = 10
200 learning.rate = 0.1
201 momentum = 0.0056
202 weight_decay = 0.0046
203  initializer  = 0.0667
204
205
206 # AND SOME HELPER FUNCTIONS:
207 mLogLoss.normalize = function(p, min_eta=1e-15, max_eta = 1.0){
208   #min_eta
209   for(ix in 1:dim(p)[2]) {
210     p[,ix] = ifelse(p[,ix]<=min_eta,min_eta,p[,ix]);
211     p[,ix] = ifelse(p[,ix]>=max_eta,max_eta,p[,ix]);
212   }
213   #normalize
214   for(ix in 1:dim(p)[1]) {
215     p[ix,] = p[ix,] / sum(p[ix,]);
216   }
217   return(p);
218 }
```

```
219
220 # helper function
221 #calculates logloss
222 mlogloss = function(y, p, min_eta=1e-15,max_eta = 1.0){
223   class_loss = c(dim(p)[2]);
224   loss = 0;
225   p = mLogLoss.normalize(p,min_eta, max_eta);
226   for(ix in 1:dim(y)[2]) {
227     p[,ix] = ifelse(p[,ix]>1,1,p[,ix]);
228     class_loss[ix] = sum(y[,ix]*log(p[,ix]));
229     loss = loss + class_loss[ix];
230   }
231   #return loss
232   return (list("loss"=-1*loss/dim(p)[1],"class_loss"=class_loss));
233 }
234
235 # mxnet specific logloss metric
236 mx.metric.mlogloss <- mx.metric.custom("mlogloss", function(label, pred){
237   p = t(pred);
238   m = mlogloss(class.ind(label),p);
239   gc();
240   return(m$loss);
241 })
242
243
244
245 #-----------------------------------------------------------------------
246 # Train the nn
247 #-----------------------------------------------------------------------
248
249
250
251 train.x <- undersample_1_set_train
252 train.y <-target_undersample_1
253
254
255 #____________
256 data = mx.symbol.Variable('data')
257 #FIRST CONVOLUITIONAL LAYER + POOLING
258 conv1 = mx.symbol.Convolution(data=data, kernel=c(3, 3), num_filter = 3)
259 relu1 = mx.symbol.Activation(data=conv1, act_type="relu")
260 pool1 = mx.symbol.Pooling(data=relu1, pool_type="max", kernel=c(2,2),
        stride=c(2,2))
261
262
```

```
263 #SECOND CONVOLUTIONAL LAYER + POOLING
264 conv2 = mx.symbol.Convolution(data=pool1, kernel=c(3,3), num_filter =
        num_filters_conv2)
265 relu2 = mx.symbol.Activation(data=conv2, act_type="relu")
266 pool2 = mx.symbol.Pooling(data=relu2, pool_type="max",kernel=c(2,2),
        stride=c(2,2))
267
268 #FLATTEN THE OUTPUT
269 flatten = mx.symbol.Flatten(data=pool2)
270
271 #FEED FULLY CONNECTED LAYER, NUMBER OF HIDDEN NODES JUST
        GEOMETRIC MEAN OF INPUT(14.5 * 14.5 * 13 = 2733.25) AND OUTPUT
        (3), sqrt(2733.25*3) = 91
272 input_previo_a_filtroconv2 <- 14.5*14.5
273 n_input <- input_previo_a_filtroconv2 * num_filters_conv2
274 num_hidden_fc1 <- round(sqrt(n_input*n_output))
275
276 fc1 = mx.symbol.FullyConnected(data=flatten, num_hidden=84)
277 relu4 = mx.symbol.Activation(data=fc1, act_type="relu")
278
279 #_____________
280
281
282 fc2 = mx.symbol.FullyConnected(data=relu4, num_hidden=3) #ESTA PARA
        CLASIFICACION
283
284 mi_softmax = mx.symbol.SoftmaxOutput(data=fc2)
285
286
287 devices <- mx.cpu()
288
289 mx.set.seed(0)
290
291 #___
292 tic <- proc.time()
293 #___
294 model <- mx.model.FeedForward.create( mi_softmax #for clasification
295                                       , X=train.x
296                                       , y=train.y
297                                       , eval.data = list("data" =
                                            set_validacion,"label" =
                                            label_set_validacion)
298                                       , ctx=devices
299                                       , num.round=num.round
300                                       , array.batch.size = 75
```

```
301                                     , learning.rate = learning.rate
302                                     , momentum = momentum
303                                     , wd=weight_decay
304                                     , eval.metric = mx.metric.mlogloss
305                                     ,  initializer =mx.init.uniform( initializer )
306                                     #, epoch.end.callback =
                                            mx.callback.save.checkpoint("modelo_guardado_ccs")
                                            #(TO SAVE MODEL AT EVERY
                                            ITERATION)
307                                     , batch.end.callback =
                                            mx.callback.log.train.metric(10)#, log)
308                                     , array.layout="columnmajor"
309 )
310
311 #___
312 print(proc.time() - tic)
313 #___
314
315
316 #-----------------------------------------------------------------
317 # Validation
318 #-----------------------------------------------------------------
319
320
321   preds <- predict(model, set_validacion,
322                    ctx = NULL,
323                    array.layout = "auto")
324
325   #WE CAN INSPECT OUR LIMITED VALIDATION SET, PROBABILITIES:
326   predicciones <- t(preds)
327   predicciones <- cbind(label_set_validacion, predicciones)
328   head(predicciones)
```

Viendo las deficiencias de esta solución, pasamos al uso de TensorFlow+Keras, del cual encontramos más información en la literatura.

Con esta solución hemos logrado buenos resultados en la competición, obteniendo valores que fuimos reduciendo a medida que mejorábamos el proceso de preprocesado (capítulo 2) y afinamiento de parámetros desde 0.95 de *logloss* hasta 0.87.

El modelo es un modelo sencillo, que no tomaba mucho tiempo de entrenamiento ya que tiene pocas capas y se entrenaban con GPU y programación

paralela de manera que el tiempo en obtener resultados medianamente aceptables no era muy elevado.

3.2.2. TensorFlow+Keras

```
1
2 #IMPORTS
3
4 def im_multi(path):
5     try:
6         im_stats_im_ = Image.open(path)
7         return [path, {'size': im_stats_im_.size}]
8     except:
9         print(path)
10        return [path, {'size': [0,0]}]
11
12 def im_stats(im_stats_df):
13     im_stats_d = {}
14     p = Pool(cpu_count())
15     ret = p.map(im_multi, im_stats_df['path'])
16     for i in range(len(ret)):
17         im_stats_d[ret[i][0]] = ret[i][1]
18     im_stats_df['size'] = im_stats_df['path'].map(lambda x: ' '.join(str(s) for s in
           im_stats_d[x]['size']))
19     return im_stats_df
20
21
22 def get_im_cv2(path):
23     img = cv2.imread(path)
24     resized = cv2.resize(img, (64, 64), cv2.INTER_LINEAR)
25     return [path, resized]
26
27 def normalize_image_features(paths):
28     imf_d = {}
29     p = Pool(cpu_count())
30     ret = p.map(get_im_cv2, paths)
31     for i in range(len(ret)):
32         imf_d[ret[i][0]] = ret[i][1]
33     ret = []
34     fdata = [imf_d[f] for f in paths]
35     fdata = np.array(fdata, dtype=np.uint8)
36     fdata = fdata.transpose((0, 3, 1, 2))
37     fdata = fdata.astype('float32')
```

```
38    fdata = fdata / 255
39    return fdata
40
41def create_model(opt_='adamax'):
42    model = Sequential()
43    model.add(Convolution2D(4, 3, 3, activation='relu', dim_ordering='th',
          input_shape=(3, 64, 64))) #use input_shape=(3, 64, 64)
44    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), dim_ordering='th'))
45    model.add(Convolution2D(8, 3, 3, activation='relu', dim_ordering='th'))
46    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), dim_ordering='th'))
47    model.add(Dropout(0.2))
48
49    model.add(Flatten())
50    model.add(Dense(12, activation='tanh'))
51    model.add(Dropout(0.1))
52    model.add(Dense(3, activation='softmax'))
53
54    model.compile(optimizer=opt_, loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])
55    return model
56
57def main():
58    os.chdir('D:\Facultad\Master\Segundo cuatrimestre\SIGE\PracticaFinal')
59    test = glob.glob("pruebaTest/*.jpg")
60    test = pd.DataFrame([[p[11:len(p)],p] for p in test], columns = ['image','path'])
61    train= glob.glob("prueba1/**/*.png")+glob.glob("prueba2/**/*.jpg")
62    train = pd.DataFrame([[p[8:14],[15:len(p)],p] for p in train], columns =
          ['type','image','path'])
63
64    train = im_stats(train)
65    train = train[train['size'] != '0 0'].reset_index(drop=True) #remove bad
          images
66    train_data = normalize_image_features(train['path'])
67
68    np.save('train.npy', train_data, allow_pickle=True, fix_imports=True)
69    np.random.seed(17)
70
71    print(len(train))
72
73    le = LabelEncoder()
74    train_target = le.fit_transform(train['type'].values)
75    print(le.classes_) #in case not 1 to 3 order
76    np.save('train_target.npy', train_target, allow_pickle=True, fix_imports=True)
77    test_data = normalize_image_features(test['path'])
78
```

```
79    np.save('test.npy', test_data, allow_pickle=True, fix_imports=True)
80
81    test_id = test.image.values
82    np.save('test_id.npy', test_id, allow_pickle=True, fix_imports=True)
83
84    train_data = np.load('train.npy')
85    train_target = np.load('train_target.npy')
86
87    x_train,x_val_train,y_train,y_val_train =
          train_test_split(train_data,train_target,test_size=0.4, random_state=17)
88
89
90    datagen = ImageDataGenerator(
91              rotation_range=180,
92          width_shift_range=0.2,
93          height_shift_range=0.2,
94          shear_range=0.2,
95          zoom_range=0.2,
96          horizontal_flip=True,
97          vertical_flip=True,
98          fill_mode='nearest')
99
100   datagen.fit(train_data)
101
102
103   model = create_model()
104   print(x_train.shape)
105   print(y_train.shape)
106   model.fit_generator(datagen.flow(x_train,y_train, batch_size=15, shuffle=True),
          nb_epoch=35, samples_per_epoch=len(x_train), verbose=20,
          validation_data=(x_val_train, y_val_train))
107
108   test_data = np.load('test.npy')
109   test_id = np.load('test_id.npy')
110
111   pred = model.predict_proba(test_data)
112   df = pd.DataFrame(pred, columns=['Type_1','Type_2','Type_3'])
113   df['image_name'] = test_id
114   df.to_csv('submission0009.csv', index=False)
115
116if __name__ == '__main__':
117   #freeze_support() # Optional under circumstances described in docs
118   main()
```

3.3. Red ya entrenada

Para probar este punto hemos seguido el script que podemos ver en [6] . En el se usa la red Inception_BN, que es el estado del arte en clasificación de la batería de imágenes ***imagenet***, para clasificar las imágenes de nuestro problema. Este problema, tiene 1000 clases correspondientes a diferentes objetos u animales y clasifica una foto en función a estas. A priori, podemos observar ya un problema y es que nuestro problema es muy técnico y limitado, las redes pre-entrenadas, en ningún caso habrán entrenado con algo ni siquiera parecido al dominio de nuestro problema por lo que esta vía queda descartada.

Igualmente, a modo de curiosidad se ha probado para ver que pasaría si intentamos clasificar nuestras imágenes con esta red. Hemos probado con cuatro imágenes distintas, para comprobar como las clasifica. Para ello siguiendo el tutorial anteriormente descrito, hemos pasado 6 imágenes con características distintas.

```
1
2  # Repeat with one of the images of the cervix dataset
3  im <- load.image("pruTescrop/Type_1/0.jpg")
4  normed <- preproc.image(im, mean.img)
5  prob <- predict(model, X = normed)
6  max.idx <- max.col(t(prob))
7  print(paste0("Predicted Top-class: ", synsets[[max.idx]]))
8
9  # Repeat with one of the images of the cervix dataset
10 im <- load.image("pruTescrop/Type_1/1168.jpg")
11 normed <- preproc.image(im, mean.img)
12 prob <- predict(model, X = normed)
13 max.idx <- max.col(t(prob))
14 print(paste0("Predicted Top-class: ", synsets[[max.idx]]))
15
16 # Repeat with one of the images of the cervix dataset
17 im <- load.image("pruTescrop/Type_2/64.jpg")
18 normed <- preproc.image(im, mean.img)
19 prob <- predict(model, X = normed)
20 max.idx <- max.col(t(prob))
21 print(paste0("Predicted Top-class: ", synsets[[max.idx]]))
22
23 # Repeat with one of the images of the cervix dataset
24 im <- load.image("pruTescrop/Type_2/913.jpg")
25 normed <- preproc.image(im, mean.img)
```

```
26 prob <- predict(model, X = normed)
27 max.idx <- max.col(t(prob))
28 print(paste0("Predicted Top-class: ", synsets[[max.idx]]))
29
30 # Repeat with one of the images of the cervix dataset
31 im <- load.image("pruTescrop/Type_3/71.jpg")
32 normed <- preproc.image(im, mean.img)
33 prob <- predict(model, X = normed)
34 max.idx <- max.col(t(prob))
35 print(paste0("Predicted Top-class: ", synsets[[max.idx]]))
36
37 # Repeat with one of the images of the cervix dataset
38 im <- load.image("pruTescrop/Type_3/1127.jpg")
39 normed <- preproc.image(im, mean.img)
40 prob <- predict(model, X = normed)
41 max.idx <- max.col(t(prob))
42 print(paste0("Predicted Top-class: ", synsets[[max.idx]]))
```

Los resultados a lo menos, son curiosos y podemos verlos en los pie de imagen de las siguientes figuras, donde se ilustra la clase asignada, y el tipo e imagen de nuestro problema. Algunos resultados son fáciles de comprender, como las que clasifica como *granadas*, por otro lado, la imagen clasificada como un *braco de Weimar* no guarda parecido ninguno.

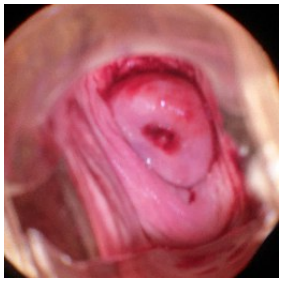

Figura 3.1: Type 1 imagen 0, clasificada como Weimaraner (perro).

Figura 3.2: Type 1 imagen 1168, clasificada como Pomegranate (granada).

Figura 3.3: Type 2 imagen 64, clasificada como Pomegranate (granada).

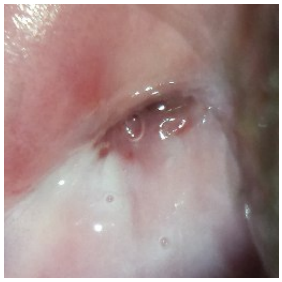

Figura 3.4: Type 2 imagen 913, clasificada como bubble, (burbuja).

Figura 3.5: Type 3 imagen 1127, clasificada como Pomegranate (granada).

Figura 3.6: Type 3 imagen 71, clasificada como Meat Shop (carnicería).

3.4. Fine Tunning

Capítulo 4

Multiclasificación y mapas de características

4.1. Multiclasificacion

4.2. Mapas de características

Capítulo 5

Conclusiones y vías futuras

En este capítulo final se estudian los resultados obtenidos a lo largo del trabajo y vías futuras para aumentar aún más el accuracy. También se complementan las conclusiones que se han ido obteniendo a lo largo del trabajo. Por tanto, podríamos resumir las conclusiones finales del trabajo en las siguientes:

5.1. Conclusiones finales

5.2. Vías futuras

Capítulo 6

ANEXO I

En este Anexo, podemos encontrar la tabla con los resultados obtenidos en las distintas entregas a Kaggle, el número de resultados no coincide al 100 % con los subidos a Kaggle, dado que a petición del profesor de teoría, Francisco Herrera, se realizó una batería de experimentos sobre el problema con el algoritmo XGBoost. Los resultados de estos experimentos fueron obviados del problema ya que no ofrecían mejora alguna sobre los resultados.

Sol	***Preprocesado***	***Algoritmos***	***Acc Test***	***Acc Training***	***Pos***
1	Imputados valores perdidos	Asumimos todos mueren	0.61	0.616	6429
2	Imputados valores perdidos	Asumimos que solo viven las mujeres	0.7655	0.7867	4781
3	Imputados valores perdidos	Todos los hombres y las mujeres de tercera clase con >20 de Fare mueren, las demás viven.	0.77990	0.8080	3259
4	Imputados valores perdidos Creada variable isChild	Ademas de lo anterior, niños de 2 clase viven.	0.78240	0.8181	2877
5	Imputados valores perdidos	Random Forest nativo de R	0.78469	0.8316	2543
6	Imputados valores perdidos añadidas variable Title, isMother, isChild	Random Forest nativo de R	0.78469	0.8316	2543
7	Imputados valores perdidos, añadida variable Title	Random Forest rparty	0.79904	0.838	1732
8	Imputados valores perdidos, añadida variable Title, eliminados outliers	Random Forest rparty	0.80383	0.8432	936
9	Imputados valores perdidos, añadidas title, familysize y familyID,	Random Forest rparty	0.8134	0.8356	473
10	Añadidas nuevas características Eliminado outliers, imputados valores perdidos	Random Forest muy ajustado y probado con distintos parámetros quedándonos con el mejor en training	0.82297	0.88431	218

Tabla 6.1: Tabla de resultados

Bibliografía

[1] Repositorio del proyecto. https://github.com/joseangeldiazg/

[2] Website de Kaggle https://www.kaggle.com

[3] Competicion en Kaggle. https://www.kaggle.com/c/intel-mobileodt-cervical-cancer-screening

[4] Tutorial MXNET. https://www.kaggle.com/miguelpm/r-mxnet-simple-tutorial

[5] Script Tensorflow. https://www.kaggle.com/marek3000/test-num-001/code

[6] Tutorial uso red preentrenada https://github.com/dmlc/mxnet/blob/master/R-package/vignettes/classifyRealImageWithPretrainedModel.Rmd