



MINERÍA DE DATO ASPECTOS AVANZADOS  
MASTER EN CIENCIA DE DATOS E INGENIERIA DE  
COMPUTADORES

## Práctica final: Deep Learning

---

**Autores**

José Ángel Díaz García, Manuel Payan Cabrera y Gerardo Fernández  
Rodríguez

**Equipo**

Equipo8

**Puntuación y Posición**

0.97500 - 10



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

Granada, Abril de 2018

# Índice general

|   |           |
|---|-----------|
| <b>1. Introducción</b>                  | <b>5</b>  |
| 1.1. Motivación . . . . .               | 5         |
| 1.2. Problema y Dataset . . . . .       | 6         |
| 1.2.1. Evaluación . . . . .             | 6         |
| 1.3. Herramientas y objetivos . . . . . | 7         |
| 1.3.1. Hardware . . . . .               | 7         |
| 1.3.2. Software . . . . .               | 7         |
| 1.3.3. Objetivos . . . . .              | 7         |
| 1.4. Organización del trabajo . . . . . | 8         |
| <b>2. Preprocesado</b>                  | <b>9</b>  |
| 2.1. Resize data . . . . .              | 9         |
| 2.2. Data Augmentation . . . . .        | 9         |
| 2.3. Filtros de ruido . . . . .         | 10        |
| <b>3. Clasificación con NN</b>          | <b>11</b> |
| 3.1. Intel Deeplearning SDK . . . . .   | 11        |
| 3.2. From Scratch . . . . .             | 12        |
| 3.2.1. Mxnet . . . . .                  | 12        |
| 3.2.2. TensorFlow+Keras . . . . .       | 20        |
| 3.3. Red ya entrenada . . . . .         | 24        |
| Práctica final: Deep Learning           | 1         |

|  |           |
|--|-----------|
| 3.4. Fine Tunning . . . . .                            | 27        |
| <b>4. Multclasificación y mapas de características</b> | <b>34</b> |
| 4.1. Multclasificación . . . . .                       | 34        |
| 4.1.1. One vs One . . . . .                            | 35        |
| 4.1.2. One vs All . . . . .                            | 35        |
| 4.1.3. OvO vs OvA . . . . .                            | 36        |
| 4.2. Mapas de características . . . . .                | 37        |
| <b>5. Conclusiones y vías futuras</b>                  | <b>38</b> |
| 5.1. Vías futuras . . . . .                            | 38        |
| 5.2. Conclusiones finales . . . . .                    | 39        |
| <b>6. ANEXO I</b>                                      | <b>40</b> |

# Índice de figuras

|   |    |
|---|----|
| 3.1. Type 1 imagen 0, clasificada como Weimaraner (perro). . . . .  | 26 |
| 3.2. Type 1 imagen 1168, clasificada como Pomegranate (granada).    | 26 |
| 3.3. Type 2 imagen 64, clasificada como Pomegranate (granada). .    | 26 |
| 3.4. Type 2 imagen 913, clasificada como bubble, (burbuja). . . . . | 27 |
| 3.5. Type 3 imagen 1127, clasificada como Pomegranate (granada).    | 27 |
| 3.6. Type 3 imagen 71, clasificada como Meat Shop (carnicería). .   | 27 |

# Índice de tablas

|   |    |
|---|----|
| 6.1. Resultados obtenidos en Kaggle . . . . . | 40 |
|---|----|

# Capítulo 1

## Introducción

Este documento presenta la memoria del trabajo realizado para la resolución de la práctica final de la asignatura ‘Minería de datos, aspectos avanzados’, enmarcada dentro del Máster en Ciencia de Datos de la universidad de Granada.

### 1.1. Motivación

La reciente incursión de las técnicas de minería de datos en actividades cotidianas o diarias constatando sus innumerables aplicaciones en diversos dominios o problemas, han propiciado que de en año en año sean más los profesionales de sectores muy dispares que deciden formarse en estas tareas que les lleven a ostentar en un futuro próximo el título de científicos de datos.

A caballo entre la minería de datos y la inteligencia artificial ha resurgido la idea de simular el comportamiento de la mente humana gracias al deep learning. El deep learning es un subconjunto dentro del machine learning, que intenta llevar a la computación la idea de aprendizaje desde el ejemplo. La idea consiste en obtener un modelo para probar ejemplos y un conjunto de reglas para modificar el modelo en el caso que se produzca un error. Para llevar a cabo este proceso se utilizan redes neuronales.

Este creciente interés ha propiciado también un caldo de cultivo perfecto para diversas plataformas online que ayudan a estos profesionales en su proceso de formación siendo una de las plataformas más conocidas Kaggle.

Esta plataforma, se basa en proponer problemas reales de minería de datos que cualquier persona interesada en la temática puede intentar resolver, en muchos casos sin grandes necesidades de cómputo o máquinas potentes. Esta plataforma evalúa los resultados aportados por cada participante y evalúa un ranking en función de diversas medidas de bondad, lo que la ha convertido también en una herramienta esencial entre los docentes del ámbito de la ciencia de datos que la usan para enfrentar a sus alumnos con problemas reales de ciencia de datos. Esta memoria, se centra por tanto en detallar el proceso llevado a cabo por los autores para resolver un problema propuesto por los profesores de la asignatura en la plataforma Kaggle. El problema en cuestión, será definido en la siguiente sección.

## 1.2. Problema y Dataset

El problema propuesto, es un problema de clasificación binaria de imágenes en el cual deberemos crear un modelo basado en redes neuronales artificiales, capaz de clasificar las imágenes correctamente. Las clases a clasificar son **armas de fuego** y **smartphones**. Para entrenar y validar el modelo disponemos de los siguientes datos:

- **train:** Tenemos dos carpetas, una con las armas de fuego y otras de smartphones. Concretamente tendremos 392 referentes a armas de fuego y 351 referentes a smartphones.
- **test:** 800 ítems mezclados de ambas clases a clasificar

### 1.2.1. Evaluación

La evaluación de la práctica es mediante la medida del Accuracy estándar, por lo que no habrá penalización entre falsos negativos y falsos positivos, como ocurriría en un problema similar por ejemplo con datos médicos.

## 1.3. 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.

### 1.3.1. Hardware

Las herramientas hardware usadas han sido los pc personales de cada uno de los miembros del equipo, en los cuales alguno carece de GPU y en los demás no se ha podido configurar correctamente las mismas por lo que el 100 % del computo ha sido llevado a cabo en CPUs.

### 1.3.2. 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 las máquinas.

- **Tensorflow:** Entorno de deeplearning sobre Python.
- **Keras:** Capa de abstracción sobre Tensorflow.
- **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.

### 1.3.3. Objetivos

Los objetivos de este trabajo podrían resumirse en los siguientes:

- Obtener un modelo predictivo fiable que dado una nueva imagen pueda predecir si es un arma o un smartphone.
- Obtener un valor de *accuracy* aceptable para escalar posiciones en la competición de Kaggle.



- Comprender y estudiar las distintas técnicas de minería de datos avanzada vista en la asignatura.

## 1.4. 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. Finalizaremos 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

Dentro del trabajo como analista de datos el apartado del preprocesamiento de los datos es una de las partes más importantes que se han de abordar. En este capítulo enunciaremos el proceso de preprocesado llevado a cabo durante la realización de la práctica.

### 2.1. Resize data

Uno de los problemas que más nos hemos encontrado en la realización de la práctica es la dimensión de los datos, ya que las imágenes cambian de forma y dimensiones por lo que en función del modelo de red neuronal usado aplicaremos uno u otro tamaño de entrada.

En este problema en concreto, el problema no es muy elevado, pero en problemas de tipo médico con imágenes de gran resolución contar con un dataset fácilmente manejable en memoria es determinante. En nuestro caso, para las primeras aproximaciones hemos redimensionado los datos a **54x54** píxeles.

### 2.2. Data Augmentation

Uno de los principales problemas en *deep learning* es la falta de datos. Para ello, pueden usarse técnicas de *data augmentation* que consisten en aplicar

ligeras transformaciones a las imágenes para conseguir un conjunto de entrenamiento mayor, pudiendo obtener de una sola imagen 5 o 6 variaciones que enriquecen enormemente el modelo. Para ello, usando las funciones propias de Keras hemos aplicado las siguientes transformaciones a las imágenes:

- **rotation\_range**: Se generan imágenes aleatorias que se rotan una cantidad de grados definidos.
- **rheight\_shift\_range**: Cambio aleatorio en la altura.
- **rwidth\_shift\_range**: Cambio aleatorio en el ancho.
- **rshear\_range**: Rango de corte.
- **rzoom\_range**: Zoom aleatorio.
- **rhorizontal\_flip**: Flip aleatorio de forma horizontal.
- **rfill\_mode**: Relleno de los puntos de la frontera.

Con todos estos cambios generamos un conjunto de imágenes más rico que nos proporciona variedad a la hora de entrenar el modelo, evitando o suavizando así el problema de no contar con muchos datos de entrada para la fase de entrenamiento, el cual sigue siendo uno de los grandes problemas a los que el deep learning se enfrenta.

## 2.3. Filtros de ruido

Además de las técnicas anteriormente descritas se han probado distintos filtros de ruido para mejorar el resultado de las imágenes. La mejor solución dentro de la aplicación de estas técnicas la hemos encontrado usando SMOOTH que se fundamenta en el suavizado de fronteras para que sea más sencillo la detección de objetos dentro de las imágenes.

# 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 ???. 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 pre-procesado 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 ?? 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 píxeles 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$Type - 1
140
141
142 set.seed(9)
143 set_validation <- ordered_images_all %>%
144   group_by(Type) %>%
145   sample_n(25) %>%
146   ungroup()
147
148
149 #-----
150 # Undersampling
151 #-----
152
153
154 set_train_unbalanced <- ordered_images_all[ordered_images_all$paciente %in%
155   setdiff(ordered_images_all$paciente, set_validation$paciente), ]
156
157 #primera muestra train undersampled:
158 set.seed(9)
159 undersample_1_set_train <- set_train_unbalanced %>%
160   group_by(Type) %>%
161   #sample_n(225) %>%
162   ungroup() %>%
163   sample_frac(1) %>%
164   sample_frac(1) #WE SHUFFLE TWICE
165
166 # dim(undersample_1_set_train)
167 # #[1] 675 12290
168 # dim(set_validation)
169 # #[1] 75 12288
170
171 # ----- train and validación labels -----
172 target_undersample_1 <- undersample_1_set_train$Type
173 label_set_validation <- set_validation$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 CONVOLUTIONAL 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),
261   stride=c(2,2))
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" =
298                                       set_validation,"label" =
299                                       label_set_validation)
300                                     , ctx=devices
301                                     , num.round=num.round
302                                     , 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)
            , 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.

### 3.2.2. TensorFlow+Keras

La solución aportada por MXNET, está lejos de ser aceptable. Por ello nos decantamos por estas librerías que a nuestro juicio ofrecen mayor versatilidad y potencia.

Con **TensorFlow+Keras** 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.

Notar como en la línea 90, volvemos a generar transformaciones sobre los datos para aumentar aun más internamente el conjunto de training y aprender mejor.

---

```

1
2#IMPORTS
3
4def 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
12def 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
19        im_stats_d[x]['size']))
19    return im_stats_df
20
21
22def get_im_cv2(path):
23    img = cv2.imread(path)
24    resized = cv2.resize(img, (64, 64), cv2.INTER_LINEAR)
25    return [path, resized]
26
27def 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
41 def create_model(opt_='adamax'):
42     model = Sequential()
43     model.add(Convolution2D(4, 3, 3, activation='relu', dim_ordering='th',
44         input_shape=(3, 64, 64))) #use input_shape=(3, 64, 64)
45     model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), dim_ordering='th'))
46     model.add(Convolution2D(8, 3, 3, activation='relu', dim_ordering='th'))
47     model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), dim_ordering='th'))
48     model.add(Dropout(0.2))
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',
55         metrics=['accuracy'])
56     return model
57
58 def main():
59     os.chdir('D:\Facultad\Master\Segundo cuatrimestre\SIGE\PracticaFinal')
60     test = glob.glob("pruebaTest/*.jpg")
61     test = pd.DataFrame([p[11:len(p)],p] for p in test, columns=['image','path'])
62     train = glob.glob("prueba1/**/*.png")+glob.glob("prueba2/**/*.jpg")
63     train = pd.DataFrame([p[8:14],[15:len(p)],p] for p in train, columns =
64         ['type','image','path'])
65
66     train = im_stats(train)
67     train = train[train['size'] != '0 0'].reset_index(drop=True) #remove bad
68         images
69     train_data = normalize_image_features(train['path'])
70
71     np.save('train.npy', train_data, allow_pickle=True, fix_imports=True)
72     np.random.seed(17)
73
74     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
116 if __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 seis imágenes distintas, para comprobar como las clasifica. Para ello siguiendo el tutorial anteriormente descrito, hemos pasado 6 imágenes con características distintas, concretamente dos de tipo 1, dos de tipo 2 y dos de tipo 3, en las cuales hemos intentado encontrar características distintas entre ellas para poder comprobar más profundamente el comportamiento de este clasificador.

---

```

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.

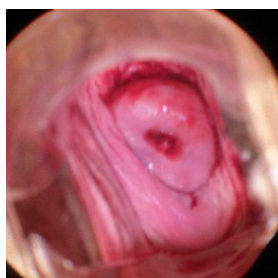


Figura 3.1: Type 1 imagen 0, clasificada como Weimaraner (perro).

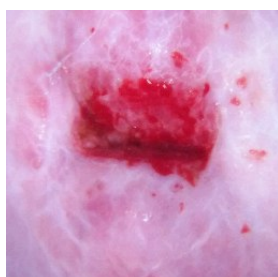


Figura 3.2: Type 1 imagen 1168, clasificada como Pomegranate (granada).



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

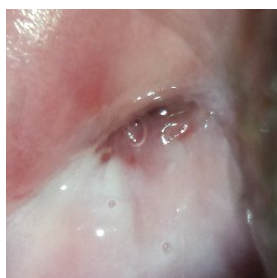


Figura 3.4: Type 2 imagen 913, clasificada como bubble, (burbuja).



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

La técnica con la que mejor resultado hemos obtenido ha sido *fine tuning*. Esta técnica, a modo de resumen, consiste en obtener una topología de red que esté en el estado del arte del reconocimiento de imágenes y adaptarla a

nuestro problema, entrenando las últimas capas con las imágenes objetivo ya que de otro modo estaríamos en una situación parecida a la vista en el punto anterior.

Estudiando distintas topologías de red para nuestro problema, finalmente nos hemos decantado por usar la red **ResNet50** [7], la cual según la literatura ofrece grandes resultados en problemas de clasificación de imágenes.

Para adaptar la red a nuestro problema, hemos reducido las imágenes a 224x224 y añadidos dos capas a la salida, una capa *Flatten* y una capa *Fully Connected* de tamaño 3, una para cada una de las posibles clases de salida de nuestro problema la activación de esta capa es de tipo *softmax*, necesaria cuando trabajamos con problemas de multclasificación, ya que nos dará los resultados en forma de las probabilidades deseadas para nuestra función de log loss.

La primera aproximación al script podemos verla a continuación. Hemos suprimido los import y algunas líneas que se dan por explicadas.

---

```

1
2def im_multi(path):
3    try:
4        im_stats_im_ = Image.open(path)
5        return [path, {'size': im_stats_im_.size}]
6    except:
7        print(path)
8        return [path, {'size': [0,0]}]
9
10def im_stats(im_stats_df):
11    im_stats_d = {}
12    p = Pool(cpu_count())
13    ret = p.map(im_multi, im_stats_df['path'])
14    for i in range(len(ret)):
15        im_stats_d[ret[i][0]] = ret[i][1]
16    im_stats_df['size'] = im_stats_df['path'].map(lambda x: ' '.join(str(s) for s in
17        im_stats_d[x]['size']))
18    return im_stats_df
19
20def get_im_cv2(path):
21    img = cv2.imread(path)
22    resized = cv2.resize(img, (64, 64), cv2.INTER_LINEAR)
23    return [path, resized]
24

```

```

25 def normalize_image_features(paths):
26     imf_d = {}
27     p = Pool(cpu_count())
28     ret = p.map(get_im_cv2, paths)
29     for i in range(len(ret)):
30         imf_d[ret[i][0]] = ret[i][1]
31     ret = []
32     fdata = [imf_d[f] for f in paths]
33     fdata = np.array(fdata, dtype=np.uint8)
34     fdata = fdata.astype('float32')
35     fdata = fdata / 255
36     return fdata
37
38 def main():
39     os.chdir('D:\Facultad\Master\Segundo cuatrimestre\SIGE\PracticaFinal')
40     test = glob.glob("pruebaTest/*.jpg")
41     test = pd.DataFrame([p[11:len(p)],p] for p in test, columns = ['image','path'])
42     train = glob.glob("prueba1/**/*.png")+glob.glob("prueba2/**/*.jpg")
43     train = pd.DataFrame([p[8:14],p[15:len(p)],p] for p in train, columns =
44         ['type','image','path'])
45
46     train = im_stats(train)
47     train = train[train['size'] != '0 0'].reset_index(drop=True) #remove bad
48         images
49     train_data = normalize_image_features(train['path'])
50
51     np.save('train.npy', train_data, allow_pickle=True, fix_imports=True)
52
53     print(len(train))
54
55     le = LabelEncoder()
56     train_target = le.fit_transform(train['type'].values)
57     print(le.classes_) #in case not 1 to 3 order
58     np.save('train_target.npy', train_target, allow_pickle=True, fix_imports=True)
59     test_data = normalize_image_features(test['path'])
60
61     np.save('test.npy', test_data, allow_pickle=True, fix_imports=True)
62
63     test_id = test.image.values
64     np.save('test_id.npy', test_id, allow_pickle=True, fix_imports=True)
65
66     x_train,x_val_train,y_train,y_val_train =
67         train_test_split(train_data,train_target,test_size=0.25, random_state=17)

```

---

```

67 datagen = ImageDataGenerator(rotation_range=0.3, zoom_range=0.3)
68
69 datagen.fit(train_data)
70
71 base_model = ResNet50(weights='imagenet', include_top=False)
72
73 x = Flatten()(base_model.output)
74 output = Dense(3, activation='softmax')(x)
75
76 model = Model(inputs=base_model.input, outputs=output)
77
78 # train only the top layers
79 for layer in base_model.layers:
80     layer.trainable = False
81
82 # compile the model
83 model.compile(optimizer='adamax', loss='sparse_categorical_crossentropy')
84 model.summary()
85
86 print('Training ... ')
87 model.fit_generator(generator=datagen.flow(x_train, y_train,
88     batch_size=15, shuffle=True),
89     validation_data=(x_val_train, y_val_train),
90     verbose=1, epochs=35, samples_per_epoch=len(x_train))
91
92 print('Predicting ... ')
93 pred = model.predict(test_data)
94
95 df = pd.DataFrame(pred, columns=['Type_1', 'Type_2', 'Type_3'])
96 df['image_name'] = test_id
97 df.to_csv('submission.csv', index=False)
98
99 if __name__ == '__main__':
100     #freeze_support() # Optional under circumstances described in docs
101     main()

```

---

Este script, ofrecía resultados muy buenos en training claramente, estaba sobreaprendiendo ya que al subir los resultados a Kaggle, este nos ofrecía valores muy malos en la competición. Igualmente, al añadir en datagen, un mayor número de modificaciones y hacer varias pruebas variando el conjunto de entrenamiento (mas o menos imágenes) y variando en número de épocas de entrenamiento, conseguimos obtener el mejor resultado con un valor de log loss en test de 0.85.

En este punto, concluimos que el problema de estas redes y cuya solución podría darnos una ventaja en la competición vendría dado por los siguientes puntos:

- Por un lado, los requisitos de memoria. Al intentar cargar todo el conjunto de training, para entrenar con imágenes extra incluidas la memoria desbordaba y aquí poca solución había posible sin perder mucha resolución por lo cual reducimos el dataset al original+transformaciones.
- Por otro lado, tenemos el problema del sobreentrenamiento. Aquí podemos usar varias técnicas como validación cruzada o algo muy extendido en deep learning, el **early stopping**, que aunque no es una solución para el sobreentrenamiento como tal puede ayudar a prevenirlo en cierta medida. Esta ha sido la solución por el cual nos decantamos.

Para implementar early stopping con Tensorflow+Keras hemos procedido de la siguiente manera. Por un lado, tal y como podemos ver en el siguiente script, guardamos el modelo en json y creamos un checkpoint con el log loss de validación. Este, guardará un archivo hdf5 cada vez que se mejore el log\_loss.

---

```

base_model = ResNet50(weights='imagenet', include_top=False)
2
3     # add a global spatial average pooling layer
4     x = GlobalAveragePooling2D()(base_model.output)
5     # add a fully-connected layer
6     x = Dense(512, activation='relu')(x)
7     # add a logistic layer
8     output = Dense(3, activation='softmax')(x)
9
10    for layer in base_model.layers:
11        layer.trainable = False
12
13    model = Model(inputs=base_model.input, outputs=output)
14
15    for layer in base_model.layers:
16        layer.trainable = False
17
18    model.compile(optimizer='adamax', loss='sparse_categorical_crossentropy')
19    model.summary()
20    model_json = model.to_json()
21

```



---

```

22 experimento_path="/Experimentos/Fine"
23
24 if not os.path.exists(experimento_path):
25     os.makedirs(experimento_path)
26
27 with open(experiment_name + "/" + "model.json","w") as json_file:
28     json_file.write(model_json)
29
30 print("Entrenando...")
31
32 filepath = + "/checkpoint-val_loss{val_loss:.5f}.hdf5"
33 checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,
34                             save_best_only=True, mode='max')
35 callbacks_list = [checkpoint]
36
37 model.fit_generator(generator=datagen.flow(x_train, y_train,
38                                           batch_size=5, shuffle=True),
39                     validation_data=(x_val_train, y_val_train),
40                     verbose=1, epochs=35, steps_per_epoch=len(x_train),
41                     callbacks=callbacks_list)

```

---

Una vez guardados estos archivos hdf5 con resultados a priori buenos, usaremos el siguiente script que carga el modelo en json, carga los pesos de ese modelo y valida nuestro conjunto de test.

---

```

1
2 test = glob.glob("pruebaTest/*.jpg")
3 test = pd.DataFrame([p[11:len(p)],p] for p in test], columns = ['image','path'])
4 test_data = normalize_image_features(test['path'])
5 np.save('test.npy', test_data, allow_pickle=True, fix_imports=True)
6 test_id = test.image.values
7 np.save('test_id.npy', test_id, allow_pickle=True, fix_imports=True)
8
9 experiment_name = "Experimentos/fine"
10
11 json_file = open(experiment_name + '/model.json', 'r')
12 model_json = json_file.read()
13 json_file.close()
14
15 model = model_from_json(model_json)
16
17 model.load_weights(experiment_name +
18                   "/CAMBIAR MANUALMENTE POREL MEJOR")

```

---

```
19 pred = model.predict(test_data)
20
21 df = pd.DataFrame(pred, columns=['Type_1', 'Type_2', 'Type_3'])
22 df['image_name'] = test_id
23 df.to_csv('submission.csv', index=False)
```

---

Esta aproximación es muy potente ya que permite dejar la máquina en computo durante largas épocas sin miedo de perder buenas soluciones ya que cuando estas mejoren se guardarán los pesos de la red y podremos comprobar como funcionan en Kaggle.

Pese a que la aproximación es buena, no conseguimos mejorar el resultado de la anterior aproximación y por falta de tiempo, no pudimos afinar esta.

En este punto finalizamos la explicación del proceso de entrenamiento y la competición como tal, ya que en el punto siguiente veremos a modo teórico a aproximaciones de multclasificación como *OvO* o *OvA*.

## Capítulo 4

# Multclasificación y mapas de características

En este capítulo estudiaremos teóricamente técnicas de multclasificación y el uso de otro tipo de clasificadores en nuestro problema.

### 4.1. Multclasificación

El problema frente al que nos encontramos, es un problema en el que una imagen a clasificar puede pertenecer a tres clases distintas, es decir, estamos ante un problema multiclase. Algunos clasificadores muy potentes, como es el caso de SVM, no pueden trabajar de manera apropiada con estos problemas ya que ofrecen su máxima potencia en problemas binarios, mismo problema que encontramos con algunos métodos de evaluación como el AUC [11] en los que es necesario disponer de una manera de traspasar el problema a un dominio binario que permita estas evaluaciones o el uso de estos algoritmos. El uso fundamental de estas técnicas es el siguiente:

- Descomponer un problema multi-clase en problemas binarios.
- Aprender un clasificador para cada subproblema.
- Agregación de las salidas en un solo clasificador.

En la literatura [8] se estudian distintos enfoques a estos problemas cada vez más exhaustivamente dado su interés en el ámbito de la investigación dentro de la ciencia de datos. En esta sección estudiaremos algunos de estos enfoques.

#### 4.1.1. One vs One

Es el enfoque más sencillo de estudiar. En este, se entrenan clasificadores acotando el problema a solo dos clases e iterando de manera que se generan clasificadores para cada una de las posibles combinaciones de clases. Con las salidas de estos clasificadores se genera una matriz de votos que se normalizará y agregará para obtener la salida final. En esta aproximación, algunos de los métodos de agregación más extendidos que pueden usarse son:

- **Voto mayoritario:** La clase ganadora es aquella que haya tenido un mayor número de respuesta positivas.
- **Voto Ponderado:** La clase ganadora es aquella cuya suma de respuestas sea la de mayor magnitud.
- **Grafo acíclico dirigido de decisión:** Se crea un árbol binario de decisión donde cada nodo representa la respuesta de un clasificador. De esta manera, cuando una clase es predicha, otra clase es descartada, por lo que es necesario descartar la pertenencia a  $C-1$  clases. La clase ganadora es aquella con la respuesta positiva en el último nodo. En los nodos superiores se colocan los clasificadores que produzcan una mayor separación entre clases, aunque no hay una evidencia sustancial de que el ordenamiento de los nodos con alguna preferencia mejore la clasificación.

#### 4.1.2. One vs All

En esta aproximación, se enfrenta una clase frente a todas las demás. Es decir, la clase en cuestión se toma como positiva y todas las demás pasan a ser negativas. Una vez hecho esto, se entrenan clasificadores para cada caso y tras ello, al igual que en la aproximación *OvO* se agregan los resultados.

Este tipo de aproximaciones tiene una clara y obvia desventaja y es que los problemas tienen que lidiar con un alto ratio de desbalanceo entre clases, lo que hace que se actualmente se estudien en conjunción a técnicas de oversampling o undersampling.

Algunas técnicas de agregacion en este punto son las siguientes:

- **Voto mayoritario:** Similar a *OvO*.
- **Uno contra todos ordenados dinámicamente:** Además de los clasificadores *OvA*, se entrena un clasificador multiclase **Näive Bayes** que ayuda a la ordenación.
- **Estrategia A&O:** Combinación de técnicas *OvO* y *OvA*.

#### 4.1.3. OvO vs OvA

Ventajas OVO:

- Problemas más sencillos.
- Problemas más pequeños.
- Computacionalmente más rápido.
- Generalmente, más preciso.

Desventajas OVO:

- Región no clasificable.
- Clasificadores no competentes.
- Crecimiento cuadrático en el número de clasificadores.

Ventajas OVA:

- Utiliza todos los ejemplos.

- No hay clasificadores no competentes.
- Agregaciones más simples
- Crecimiento lineal en el número de clasificadores.

Desventajas OVA:

- Problemas no balanceados..
- Problemas más complejos.
- Computacionalmente más costosos.

## 4.2. Mapas de características

Una de las últimas vías de investigación con redes neuronales pasa por utilizar las redes para obtener los mapas de características de una imagen y utilizar otros clasificadores distintos de redes neuronales para clasificar estos. Esta vía, es muy avanzada y hemos entrado a modo teórico probando el script dado en prácticas en el cual se usa un RandomForest para este objetivo.

Tras probarlo, los resultados son muy malos, debido a que RandomForest no trabaja bien con este tipo de datos sino que necesita datos categóricos. Por ello, el uso de XGBOOST (estado del arte) el cual utiliza datos reales para su entrenamiento ofrecería grandes resultados en esta forma de atacar el problema.

Otro clasificador interesante, podría ser las máquinas de soporte vectorial, junto con las cuales y Xgboost podríamos desarrollar un ensemble que podría obtener muy buenos resultados, al menos teóricamente.

# 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.

### 5.1. Vías futuras

Respecto a las vías futuras, pese a que algunas ya han sido comentadas a lo largo de la memoria, cabe reunir las en este último capítulo.

Principalmente, destacar la necesidad de disponer de máquinas más potentes a la hora de procesar esta gran cantidad de imágenes para poder disponer de todo el dataset sin necesidad de redimensionar y por consiguiente pérdida de calidad.

Respecto a las mejoras en la implementación, ensembles de diversas topologías de redes neuronales con *fine tuning* y prevención del *overfitting* serían muy probablemente la mejor opción para atacar el problema. Estas soluciones han sido estudiadas en la literatura [12] pero no hemos conseguido implementarlas ya que son complejas y aún residen casi en su totalidad en el ámbito de investigación de las mismas.

## 5.2. Conclusiones finales

Como conclusión que aúna todas las demás podemos concluir que el problema era complejo y extenso. Se pedían probar un gran número de técnicas y además conseguir buenos resultados en la competición, con lo cual, esto ha causado que se haya entrado por encima en algunas de las mismas con el fin de profundizar más en otras que a priori puedan parecer mejores.

Respecto a las características del problema, muchas de las soluciones vistas en clase no ofrecían buenos resultados llegando incluso algunas a no poder ejecutarse por falta de memoria, como MXNET que podía utilizar poco más de 1GB de imágenes. Por tanto, un problema que necesita de poder procesar una gran cantidad de imágenes, donde las herramientas disponibles para su procesamiento son tan limitadas (ordenadores personales), difícilmente podrá obtener buenos resultados.

Por otro lado, cabe destacar la potencia de las librerías de deep learning, para este tipo de problemas. Con muy poco código y con conocimientos de programación básicos, se pueden lograr grandes resultados, aunque llegar a utilizar estas herramientas nivel experto puede llegar a ser una ardua labor.

Para finalizar las conclusiones, queremos profundizar un poco en lo trascendental del problema. Estamos trabajando con datos de pacientes médicos en los que una buena solución de machine learning puede suponer salvar vidas, constatando por tanto la importancia que la ciencia de datos está tomando en la totalidad de sectores trascendiendo en algunos casos, como este mismo, de lo meramente informático u económico-empresarial.



# Capítulo 6

## ANEXO I

En este Anexo, podemos encontrar la tabla con los resultados obtenidos en las distintas entregas a Kaggle. Hay algunos parámetros no calculados debido a que algunos scripts usados no fueron desarrollados para mostrar el LogLoss por ejemplo en training, o las entregas a Kaggle que ofrecían peores resultados que las anteriores las cuales no ofrecen por tanto ranking.

| <i>Preprocesado</i>                                   | <i>Imágenes</i>   | <i>Tamaño</i> | <i>Tecnología</i> | <i>LogLossTest</i> | <i>LogLossTrain</i> | <i>Posición</i> | <i>batch</i> | <i>Epoch</i> | <i>Nº imágenes</i> |
|---|-------------------|---------------|-------------------|--------------------|---------------------|-----------------|--------------|--------------|--------------------|
| Ninguno   | Basic Train       | 64X64         | Tensorflow+Keras  | 0.99971            | 0.9915              | 488             | 500          | 10           | 1481               |
| Ninguno   | Basic Train       | 64X64         | Tensorflow+Keras  | 0.97718            | 0.9605              | 460             | 15           | 10           | 1481               |
| Flip+Flip+Rotacion+Traslacion                         | Basic Train       | 64X64         | Tensorflow+Keras  | 0.96671            | 0.9826              | 424             | 500          | 10           | 7405               |
| Ninguno   | Basic Train+Extra | 64X64         | Tensorflow+Keras  | 0.95141            | 0.9871              | 392             | 500          | 10           | 42425              |
| Ninguno   | Basic Train       | 64X64         | Tensorflow+Keras  | 0.93924            | 0.9277              | 362             | 10           | 10           | 1481               |
| Flip+Flip+Rotacion+Traslacion                         | Basic Train       | 64X64         | Tensorflow+Keras  | 0.95473            | 0.9333              | 366             | 10           | 10           | 7405               |
| Flip+Flip+Rotacion+Traslacion                         | Basic Train+Extra | 170X128       | Tensorflow+Keras  | 0.96125            | 0.9830              | -               | 10           | 10           | 42425              |
| Ninguno   | Basic Train       | 170X128       | Tensorflow+Keras  | 0.99048            | 1.0024              | -               | 500          | 10           | 1481               |
| Ninguno   | Basic Train       | 170X128       | Tensorflow+Keras  | 1.02013            | 0.9372              | -               | 10           | 10           | 1481               |
| Flip+Flip+Rotacion+Traslacion                         | Basic Train       | 64X64         | Tensorflow+Keras  | 0.95592            | -                   | -               | 10           | 10           | 42425              |
| crop+reducir  | Basic Train       | 64X64         | Tensorflow+Keras  | 0.87921            | -                   | 286             | 10           | 35           | 1481               |
| crop+reducir  | Basic Train       | 32X32         | Tensorflow+Keras  | 0.88253            | -                   | -               | 15           | 35           | 1481               |
| crop+reducir+Flip+Flip+Rotacion+Traslacion            | Basic Train       | 128X128       | Tensorflow+Keras  | 0.86259            | -                   | 264             | 15           | 35           | 7405               |
| crop+reducir+Flip+Flip+Rotacion+Traslacion            | Basic Train       | 128X128       | Tensorflow+Keras  | 0.92413            | -                   | -               | 15           | 100          | 7405               |
| crop+reducir  | Basic Train+Extra | 128X128       | Tensorflow+Keras  | 0.89062            | -                   | -               | 15           | 10           | 42425              |
| crop+reducir+Balanceo+Flip+Flip+Rotacion+Traslacion   | Basic Train       | 128X128       | Tensorflow+Keras  | 0.92330            | -                   | -               | 15           | 10           | 21943              |
| crop+reducir+Flip+Flip+Rotacion+Traslacion            | Basic Train+Extra | 64X64         | Tensorflow+Keras  | 0.85521            | -                   | 259             | 15           | 35           | 42425              |
| crop+reducir+Flip+Flip+Rotacion+Traslacion            | Basic Train+extra | 64X64         | Tensorflow+Keras  | 0.85866            | -                   | -               | 15           | 50           | 42425              |
| crop+reducir+Flip+Flip+Rotacion+Traslacion            | Basic Train+extra | 64X64         | Tensorflow+Keras  | 0.88781            | -                   | -               | 15           | 20           | 42425              |
| FineTuning+crop+reducir+Flip+Flip+Rotacion+Traslacion | Basic Train       | 224x224       | Tensorflow+Keras  | 0.85985            | -                   | 262             | 15           | 20           | 7405               |

Tabla 6.1: Resultados obtenidos en Kaggle



# 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 pre-entrenada <https://github.com/dmlc/mxnet/blob/master/R-package/vignettes/classifyRealImageWithPretrainedModel.Rmd>
- [7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition.
- [8] Mikel Galar, Alberto Fernández, Edurne Barrenechea, Humberto Bustince, Francisco Herrera, An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes, Pattern Recognition, Volume 44, Issue 8, 2011, Pages 1761-1776, ISSN 0031-3203.
- [9] <http://www.tamps.cinvestav.mx/~wgomez/diapositivas/RP/Clase11.pdf>
- [10] [http://www.aepia.org/aepia/files/EVIA/TransparenciasEVIA2016/Sanz\\_Galar\\_Fernandez\\_EVIA2016.pdf](http://www.aepia.org/aepia/files/EVIA/TransparenciasEVIA2016/Sanz_Galar_Fernandez_EVIA2016.pdf)

- [11] Desarrollo de un módulo de evaluación probabilística. [http://joseangeldiazg.github.io/tfg\\_roc\\_keel/](http://joseangeldiazg.github.io/tfg_roc_keel/)
- [12] Jianjun Liu Shengping Xia Weidong Hu Wenxian Yu. Weights Updated Voting for Ensemble of Neural Networks Based Incremental Learning. International Symposium on Neural Networks. Advances in Neural Networks ? ISSN 2009 pp 661-669