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

Índice de figuras

Índice de tablas

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 match 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 [2].

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 4.

Los resultados obtenidos en la competición de Kaggle pueden encontrarse en el Anexo de la sección **??**.

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** pixeles.

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 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 [**?**] y la de TensorFlow de este otro [**?**], 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 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.

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
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
41  def 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
57  def 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
```

```
    df.to_csv('submission0009.csv', index=False)

if __name__ == '__main__':
    #freeze_support() # Optional under circumstances described in docs
    main()
```

3.3. Red ya entrenada

Para probar este punto hemos seguido el script que podemos ver en [**?**] . 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 cueles hemos intentado encontrar características distintas entre ellas para poder comprobar más profundamente el comportamiento de este clasificador.

```
# Repeat with one of the images of the cervix dataset
im <- load.image("pruTescrop/Type_1/0.jpg")
normed <- preproc.image(im, mean.img)
prob <- predict(model, X = normed)
max.idx <- max.col(t(prob))
print(paste0("Predicted Top-class: ", synsets[[max.idx]]))

# Repeat with one of the images of the cervix dataset
im <- load.image("pruTescrop/Type_1/1168.jpg")
normed <- preproc.image(im, mean.img)
prob <- predict(model, X = normed)
max.idx <- max.col(t(prob))
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.

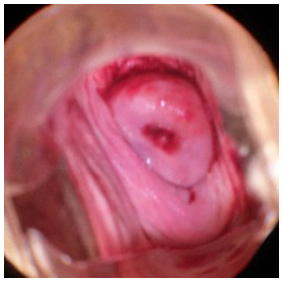

Figura 3.1: Type 1 imagen 0, clasificada como Weimaraner (perro).

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

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

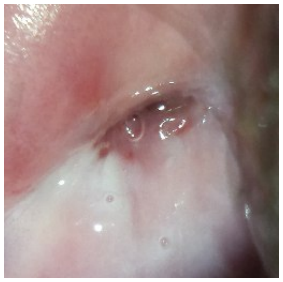

Figura 3.4: Type 2 imagen 913, clasificada como bubble, (burbuja).

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*** [**?**], 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 multiclasificación, 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 lineas 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
          im_stats_d[x]['size']))
17    return im_stats_df
18
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 =
           ['type','image','path'])
44
45     train = im_stats(train)
46     train = train[train['size'] != '0 0'].reset_index(drop=True) #remove bad
           images
47     train_data = normalize_image_features(train['path'])
48
49     np.save('train.npy', train_data, allow_pickle=True, fix_imports=True)
50
51     print(len(train))
52
53     le = LabelEncoder()
54     train_target = le.fit_transform(train['type'].values)
55     print(le.classes_) #in case not 1 to 3 order
56     np.save('train_target.npy', train_target, allow_pickle=True, fix_imports=True)
57     test_data = normalize_image_features(test['path'])
58
59     np.save('test.npy', test_data, allow_pickle=True, fix_imports=True)
60
61     test_id = test.image.values
62     np.save('test_id.npy', test_id, allow_pickle=True, fix_imports=True)
63
64     x_train,x_val_train,y_train,y_val_train =
           train_test_split(train_data,train_target,test_size=0.25, random_state=17)
65
66
```

```
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
99if __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)

 # add a global spatial average pooling layer
x = GlobalAveragePooling2D()(base_model.output)
# add a fully-connected layer
x = Dense(512, activation='relu')(x)
# add a logistic layer
output = Dense(3, activation='softmax')(x)

for layer in base_model.layers:
    layer.trainable = False

model = Model(inputs=base_model.input, outputs=output)

for layer in base_model.layers:
    layer.trainable = False

model.compile(optimizer='adamax', loss='sparse_categorical_crossentropy')
model.summary()
model_json = model.to_json()

```

```
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,
        save_best_only=True, mode='max')
34  callbacks_list = [checkpoint]
35
36  model.fit_generator(generator=datagen.flow(x_train, y_train,
37                              batch_size=5, shuffle=True),
38                              validation_data=(x_val_train, y_val_train),
39                              verbose=1, epochs=35, steps_per_epoch=len(x_train),
                                    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 +
        "/CAMBIARMANUALMENTEPORELMEJOR")
18
```

```
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 multiclasificación como *OvO* o *OvA*.

Capítulo 4

Conclusiones y vías futuras

En este capítulo final se estudian los resultados obtenidos a lo largo del trabajo y así como las vías futuras para aumentar aún más el accuracy. También se complementan las conclusiones que se han ido obteniendo a lo largo del trabajo.

4.1. Vías futuras

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

Principalmente, destacar la necesidad de disponer de máquinas más potentes o al menos con capacidad de procesado con GPU a la hora de procesar esta gran cantidad de imágenes con redes neuronales para poder ademas de mejorar los tiempos de ejecución, poder tener las imágenes con tamaños completos sin perdida de calidad.

Respecto a las mejoras en la implementación, ensembles de diversas topologías de redes neuronales con *fine tunning* y prevención del *overfitting* serían muy probablemente la mejor opción para atacar el problema. Estas soluciones han sido estudiadas en la literatura [5] pero no hemos conseguido implementarlas ya que son complejas y aún residen casi en su totalidad en el ámbito de investigación de las mismas.

4.2. Conclusiones finales

Como conclusión que aúna todas las demás podemos concluir que el problema era ciertamente complejo, y aunque permitia buenos resultados rápidamente mejorarlos iba subiendo en dificultad. Las redes neuronales y el deep learning pone a nuestra disposición una gran cantidad de técnicas (fine tunning, transfer learning?) que han sido descritas a lo largo de la memoria y las cuales se han intentado probar a modo experimental. Si sumamos esto a la necesidad de conseguir buenos resultados en la competición, creemos que el tiempo asignado para dicha práctica debería de haber sido algo más extenso lo que habría facilitado aún más la asimilación de los conceptos de la asignatura.

Respecto a las características del problema, muchas de las soluciones vistas no ofrecían buenos resultados llegando incluso algunas a no poder ejecutarse por falta de memoria. Por tanto, un problema que necesita de poder procesar imágenes, donde las herramientas disponibles para su procesado son tan limitadas (ordenadores personales), difícilmente podrá obtener buenos resultados, o al menos no todos los buenos resultados que podrían alcanzarse con equipos más potentes por lo que el sesgo entre soluciones que hayan podido hacer uso de GPU y otras que no, es elevado.

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 reales de identificación de armas y su comparación con otros objetos similares como puede ser el caso de smartphones. La potencia y uso de problemas reales del ámbito de la seguridad ha quedado constatada como es el caso del estudio de Olmos, Tabik y Herrera [6] que que ha recibido prestigiosos premios internacionales y constata la necesidad de estas soluciones en la sociedad actual.

Bibliografía

[1] Repositorio del proyecto. https://github.com/joseangeldiazg/guns-smartphones-classification

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

[3] Competicion en Kaggle. https://www.kaggle.com/c/pistolas-vs-smartphones-con-deep-learning

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE, November 1998.

[5] 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* ISNN 2009 pp 661-669

[6] R Olmos, S Tabik, F Herrera. Automatic Handgun Detection Alarm in Videos Using Deep Learning *Neurocomputing*, 275, 66-72