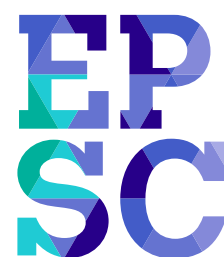




**ESCUELA POLITÉCNICA  
SUPERIOR DE CÓRDOBA**  
Universidad de Córdoba



**TRABAJO FIN DE GRADO**

**Grado en Ingeniería Informática**

# **CLASIFICACIÓN DE MELANOMAS A TRAVÉS DE REDES NEURONALES RESIDUALES**

**MANUAL DE CÓDIGO**

Autor

**D. Juan José Méndez Torrero**

Directores

**Dr. Pedro Antonio Gutiérrez Peña**

**Dr. Javier Sánchez Monedero**

**Septiembre, 2019**



UNIVERSIDAD DE CÓRDOBA





# Índice general

Índice general	I
Listings	III
Índice de tablas	V
<b>1. Introducción</b>	<b>1</b>
<b>2. Características del sistema</b>	<b>3</b>
2.1. Notación de la implementación . . . . .	3
2.2. Estructura de la documentación . . . . .	4
<b>3. Documentación del código</b>	<b>5</b>
3.1. Clase PrepareData . . . . .	5
3.1.1. Especificación de la clase <i>PrepareData</i> . . . . .	5
3.1.2. Fichero <i>PrepareData.py</i> . . . . .	6
3.2. Clase PrepareTrainTest . . . . .	9
3.2.1. Especificación de la clase <i>PrepareTrainTest</i> . . . . .	9
3.2.2. Fichero <i>PrepareTrainTest.py</i> . . . . .	10
3.3. Clase ResNet . . . . .	13
3.3.1. Especificación de la clase <i>ResNet</i> . . . . .	13
3.3.2. Fichero <i>ResNet.py</i> . . . . .	15
3.4. Clase AuxFunctions . . . . .	21
3.4.1. Especificación de la clase <i>AuxFunctions</i> . . . . .	21
3.4.2. Fichero <i>AuxFunctions.py</i> . . . . .	23
3.5. Fichero Main . . . . .	26



# Listings

3.1. Código de la clase PrepareData . . . . .	6
3.2. Código de la clase PrepareTrainTest . . . . .	10
3.3. Código de la clase ResNet . . . . .	15
3.4. Código de la clase AuxFunctions . . . . .	23
3.5. Código del fichero Main . . . . .	26



# Índice de tablas

3.1. Especificación de las variables y métodos de la clase PrepareData	6
3.2. Especificación de las variables y métodos de la clase Prepare- TrainTest . . . . .	10
3.3. Especificación de las variables y métodos de la clase ResNet .	15
3.4. Especificación de las variables y métodos de la clase AuxFunction	23

# Capítulo 1

## Introducción

Este documento es el Manual de Código del Trabajo de Fin de Grado **Clasificación de melanomas a través de redes neuronales residuales**, en el cual se expondrá una descripción detallada de todos los elementos que componen el sistema. El objetivo de este documento es el de servir a otros programadores como base para poder así realizar futuras mejoras del mismo. Este documento estará dividido en varias partes, las cuales incluirán las clases creadas para la creación del mismo.

El objetivo de este proyecto es la creación de un modelo de red neuronal profunda para la clasificación de tipos de lesiones en la piel, ya sean *malig-nas* o *benignas*. Para la creación del mismo, se ha utilizado el lenguaje de programación *Python*, junto con las librerías *Tensorflow* y *Keras*, las cuales son explicadas en el Manual Técnico de este proyecto. La razón por la que hemos decidido utilizar este lenguaje para la realización de este proyecto es porque nos facilita la tarea de creación y modificación de vectores y matrices, y como ya sabemos, las imágenes son consideradas como matrices, con lo que este lenguaje de programación es el ideal para este tipo de proyectos.

En este documento, como ya hemos dicho, se expondrán las clases creadas junto con la organización utilizada para la creación del modelo de red neuronal residual. Además, de esto, se explicará la funcionalidad de cada una de las clases haciendo énfasis en cada una de las características que presentan.





# Capítulo 2

## Características del sistema

### 2.1. Notación de la implementación

Con respecto a la notación seguida durante la codificación, hemos intentado seguir la estudiada a lo largo de la carrera. Es decir, cómo definir correctamente cada una de las variables, funciones y clases que pertenecen a cada parte del proyecto. En cuanto al código del mismo, se seguirán las siguientes notaciones:

- Las variables serán siempre definidas en minúscula. En el caso de una palabra compuesta, ésta irá separada por `_`: *batch\_size*.
- Las variables serán definidas de tal manera que el nombre de éstas pueda identificar perfectamente cuál será el uso de la misma.
- El nombre de las clases comenzarán en mayúscula. Si se da el caso de que el nombre es compuesto, las dos palabras utilizadas comenzarán con mayúscula: *AuxFunctions*.
- Las librerías utilizadas en cada uno de los ficheros estarán definidas al comienzo de los mismos.
- El nombre de las funciones creadas deberán de comenzar con minúscula, utilizando para separar dos palabras, o bien `_`, o comenzando la segunda palabra con mayúscula: *train\_model* o *trainModel*.

Todas estas notaciones siguen la guía de estilos *PEP 8* del lenguaje de programación *Python*.

## 2.2. Estructura de la documentación

Para estructurar este documento, utilizaremos las secciones para ir explicando cada uno de los ficheros creados para este proyecto. En cada sección se realizará una explicación mediante tablas de la funcionalidad de la clase, junto con la funcionalidad de las variables y funciones utilizadas en la misma. Finalmente, cada sección terminará con el código *Python* que ha sido utilizado para la creación de cada una de las clases.

# Capítulo 3

## Documentación del código

### 3.1. Clase PrepareData

Esta sección especifica la clase *PrepareData*, la cual se encargará de dividir las imágenes en tipo maligno o tipo benigno según la etiqueta de cada una. Esta especificación puede ser observada en la Tabla 3.1.

#### 3.1.1. Especificación de la clase *PrepareData*

Nombre de la clase	PrepareData	
Descripción	Esta clase será la encargada de separar los datos descargados en dos tipos, las imágenes de lesiones benignas y las imágenes de lesiones malignas	
Datos		
path_images	String	Identifica la ruta en la que se encuentran las imágenes para su clasificación inicial.

<b>main_path</b>	String	Identifica la ruta en donde se guardarán los resultados de esta clasificación.
<b>Métodos</b>		
<b>createHD5PY</b>	Guarda las imágenes encontradas en el directorio <i>path_images</i> en el directorio <i>main_path</i> . El formato de la salida es <i>hdf5</i> , en el cual se encontrarán tanto las imágenes encontradas divididas según el tipo de lesión.	
<b>readDataH5PY</b>	Lee las imágenes guardadas en el fichero <i>hdf5</i> del directorio <i>main_path</i> . Esta función devuelve en forma de array las imágenes de tipo benigno y, en otro, las imágenes de tipo maligno.	

Tabla 3.1: Especificación de las variables y métodos de la clase PrepareData

### 3.1.2. Fichero *PrepareData.py*

```

1  """
2      Project: Melanoma recognition
3      Author: Juan Jos  M ndez Torrero
4      File: PrepareData.py
5      Program: File to prepare the images and save them onto a
        hdf5 file
6  """
7  import numpy as np
8  import h5py
9  import os
10 import json
11
12 from tqdm import tqdm # This library is used to know the
        percentage of images already classified and saved
13 """
14     Name: PrepareData
15 
```

```

16     Description: This class has been created in order to
17     classify the images before the training.
18     """
19
20 class PrepareData:
21     def __init__(self, path_images, main_path):
22         self.path_images = path_images
23         self.main_path = main_path
24
25     """
26     Name: createH5PY
27
28     Inputs: - path_images: Path of the images.
29             - path_datasets: Path where it is going to
30             save the images.
31
32     Returns: None.
33
34     Description: This function classifies the images into
35     malignant and benign. After that, creates a hdf5 file in
36     order to save them.
37     """
38     def createH5PY(self):
39         files = []
40         with h5py.File(self.path_datasets + "dataset.hdf5", "
41 w") as hdf:
42             benign_group = hdf.create_group("benign_images")
43             malignant_group = hdf.create_group("
44 malignant_images")
45             print("Creating h5py file...")
46             # Read all images from directory
47             for r, d, f in os.walk(self.path_images):
48                 for file in tqdm(f):
49                     if ".json" in file:
50                         paths = os.path.join(r, file)
51                         with open(paths) as json_file:
52                             data_json = json.load(json_file)
53                             if (data_json["meta"]["clinical"]
54                                 ["benign_malignant"] == "
55 malignant"):
56                                 data = image.load_img(paths
57 [: -4] + ".jpg",
58
59 target_size=(512, 512))
60
61                                 malignant_group.
62 create_dataset(
63
64                                     file[: -4] + ".jpg",
65                                     data=data,

```

```

55             compression="gzip")
56         elif (data_json["meta"]["clinical
    "]
57             ["benign_malignant"] == "
benign"):
58             data = image.load_img(paths
[: -4] + ".jpg",
59             target_size=(512, 512))
60             benign_group.create_dataset(
file[: -4] + ".jpg",
61             data=data,
62             compression="gzip")
63
64         """
65         Name: readDataH5PY
66
67         Inputs: - path: Path of the hdf5 file.
68
69         Returns: - malignant_images: An array with all the
malignant images.
70                  - benign_images: An array with the benign
images.
71
72         Description: This function reads the hdf5 file and
returns two array with the different types of images.
73         """
74         def readDataH5PY(self):
75             malignant_images = []
76             benign_images = []
77             with h5py.File(self.main_path + "dataset.hdf5", "r")
as hdf:
78                 malignant_items = list(hdf.get("malignant_images"
).items())
79
80                 print("-" * 40)
81                 print("Getting malignant images...")
82                 print("-" * 40)
83
84                 for items in tqdm(malignant_items):
85                     malignant_images.append(
86                         np.array(hdf.get("malignant_images").get(
items[0])))
87
88                 print("-" * 40)
89                 print("Getting benign images...")
90                 print("-" * 40)

```

```

91         benign_items = list(hdf.get("benign_images").
92 items())
93         for items in tqdm(benign_items):
94             benign_images.append(
95                 np.array(hdf.get("benign_images").get(
96 items[0])))
97         return malignant_images, benign_images

```

Listing 3.1: Código de la clase PrepareData

## 3.2. Clase PrepareTrainTest

Esta sección especifica la clase *PrepareTrainTest*, la cual se encargará de dividir las imágenes de lesiones malignas y benignas ya etiquetadas en los distintos conjuntos necesarios para el entrenamiento de la red (train, test y validación). Esta especificación puede ser observada en la Tabla 3.2.

### 3.2.1. Especificación de la clase *PrepareTrainTest*

Nombre de la clase	PrepareTrainTest	
Descripción	Esta clase se encargará de separar los datos anteriormente separados en tres partes, una para el entrenamiento ( <i>train</i> ), una para los test ( <i>test</i> ) y otra para la validación ( <i>validation</i> ). Cada una de estas partes, tendrá una parte que contendrá las imágenes y otra que contendrá la etiqueta de las mismas.	
Datos		
main_path	String	Esta variable guarda la ruta en la que las imágenes para el entrenamiento, junto con su correspondiente etiqueta, son guardadas



<b>file_name</b>	String	Nombre del fichero en donde serán guardadas las imágenes con sus etiquetas.
<b>Métodos</b>		
<b>createTrainTestH5PY</b>	Esta función es la encargada de primero, agrupar las imágenes para separarlas en train, test y validación. Una vez ha hecho hecho, transforma las etiquetas en un array categórico y, finalmente, guarda esos vectores en el directorio <i>main_path</i> con el nombre <i>file_name</i> .	
<b>readDataH5PY</b>	Esta función se encarga de leer el fichero <i>file_name</i> y devolver los vectores correspondientes a cada una de las variables del entrenamiento.	

Tabla 3.2: Especificación de las variables y métodos de la clase PrepareTrainTest

### 3.2.2. Fichero *PrepareTrainTest.py*

```

1  """
2      Project: Melanoma recognition
3      Author: Juan Jos  M ndez Torrero
4      File: PrepareTrainTest.py
5      Program: File that creates the train test file in order
6              to train the model with
7  """
8  import h5py
9  from tqdm import tqdm
10 from sklearn.model_selection import train_test_split
11 import numpy as np
12 from keras.utils import np_utils
13 """
14     Name: PrepareTrainTest

```

```

15
16     Description: This class has been created in order split
17     the input data into train, test and validation for the
18     training.
19     """
20 class PrepareTrainTest:
21     def __init__(self, main_path, file_name):
22         self.main_path = main_path
23         self.file_name = file_name
24
25         """
26         Name: createTrainTestH5PY
27
28         Inputs: - path: Path of the hdf5 file.
29                 - name_file: String for the name of the hdf5
30                 file that keeps the train, test and validation data.
31                 - malignant_images: Array with the malignant
32                 images.
33                 - benign_images: Array with the benign images
34                 .
35
36         Returns: None.
37
38         Description: This function splits the recieved data
39         into train, test and validation. At the end, it saves the
40         data into a hdf5 file.
41         """
42     def createTrainTestH5PY(self, malignant_images,
43                             benign_images):
44         print("-" * 40)
45         print("Splitting data into train, test and validation
46         ...")
47         print("-" * 40)
48         # Concatenate all data and create labels
49         all_data = np.vstack((malignant_images, benign_images
50                                ))
51         labels = np.concatenate(
52             (np.zeros(len(malignant_images)), np.ones(len(
53                 benign_images))))
54
55         # Do train test split in order to get the training,
56         test and validation data
57         X_train, X_test, y_train, y_test = train_test_split(
58             all_data,
59             labels,

```

```

49     test_size=0.4,
50
51     stratify=labels)
52     X_val, X_test, y_val, y_test = train_test_split(
53         X_test,
54         y_test,
55         test_size=0.5,
56         stratify=y_test)
57
58     # Make labels categorical
59     y_train = np_utils.to_categorical(y_train)
60     y_test = np_utils.to_categorical(y_test)
61     y_val = np_utils.to_categorical(y_val)
62
63     print("-" * 40)
64     print("Creating h5py train test file...")
65     print("-" * 40)
66     dataset_output_path = self.main_path + self.file_name
67     with h5py.File(dataset_output_path, "w") as hdf:
68         hdf.create_dataset("X_train", data=X_train,
69             compression="gzip")
70         hdf.create_dataset("y_train", data=y_train,
71             compression="gzip")
72         hdf.create_dataset("X_test", data=X_test,
73             compression="gzip")
74         hdf.create_dataset("y_test", data=y_test,
75             compression="gzip")
76         hdf.create_dataset("X_val", data=X_val,
77             compression="gzip")
78         hdf.create_dataset("y_val", data=y_val,
79             compression="gzip")
80
81     """
82     Name: readDataH5PY
83
84     Inputs: - path: Path of the hdf5 file.
85             - name_file: String for the name of the hdf5
86             file that keeps the train, test and validation data.
87
88     Returns: - dataset: A dictionary that keeps the value
89             of the train, test and validation data.
90
91     Description: This function reads the data for the
92     training from a hdf5 file and returns a dictionary that
93     keeps them.

```

```

82     """
83     def readDataH5PY(self):
84         print("-" * 40)
85         print("Reading h5py train test file...")
86         print("-" * 40)
87         dataset = h5py.File(self.main_path + self.file_name,
88 "r")
89         return (
90             dataset["X_train"][()],
91             dataset["X_test"][()],
92             dataset["X_val"][()],
93             dataset["y_train"][()],
94             dataset["y_test"][()],
95             dataset["y_val"][()],
96         )

```

Listing 3.2: Código de la clase PrepareTrainTest

### 3.3. Clase ResNet

Esta sección especifica la clase *ResNet*. Esta clase es la encargada de declarar las funciones necesarias para la construcción, entrenamiento y evaluación del modelo de red neuronal residual profunda. Esta especificación puede ser observada en la Tabla 3.3.

#### 3.3.1. Especificación de la clase *ResNet*

Nombre de la clase	ResNet
Descripción	Esta clase contiene una serie de métodos que nos permitirán construir el modelo de red, entrenarlo y finalmente evaluarlo.
Datos	
lr	Decimal Identifica el ratio de aprendizaje utilizado por la red.

<b>opt</b>	Entero	Identifica el tipo de optimizador que la red utilizará para compilar el modelo. Su valor será 0, 1 o 2.
<b>batch_size</b>	Entero	Identifica el tamaño del lote que se entrenará cada iteración.
<b>epochs</b>	Entero	Identifica el máximo número de iteraciones que nuestro modelo entrenará.
<b>Métodos</b>		
<b>trainModel</b>	Esta función realizará el entrenamiento de nuestra red una vez se haya compilado con los parámetros de entrada. Nos devolverá el historial del mismo para su posterior uso en la creación de la matriz de confusión.	
<b>evaluateModel</b>	Esta función consiste en la evaluación de nuestro modelo una vez entrenado. Esta función nos mostrará por pantalla el valor de precisión y pérdida según los datos utilizados para el test.	
<b>identityBlock</b>	Esta función ha sido creada para simular el atajo hablado en el Capítulo ??.	
<b>convolutionalBlock</b>	Esta función ha sido creada para simular unir el atajo creado por la anterior función y el flujo principal de la red.	

<b>buildModel</b>	Esta función es la encargada de unir todo el flujo de capas de nuestra red neuronal residual profunda. Además, nos mostrará por pantalla un resumen de las capas utilizadas y devolverá el modelo para su uso en el resto de funciones.
-------------------	---

Tabla 3.3: Especificación de las variables y métodos de la clase ResNet

### 3.3.2. Fichero *ResNet.py*

```

1  """
2      Project: Melanoma recognition
3      Author: Juan Jos  M ndez Torrero
4      File: ResNet.py
5      Program: File that creates the model and trains it with
6               the images already processed
7  """
8  import tensorflow as tf
9  import numpy as np
10
11 import keras
12 from keras.layers import (
13     Dense,
14     Activation,
15     Flatten,
16     Conv2D,
17     MaxPooling2D,
18     BatchNormalization,
19     ZeroPadding2D,
20     Input,
21     AveragePooling2D,
22 )
23 from keras import Model
24 from keras import optimizers
25 from keras import losses
26 from keras import layers
27
28 """
29     Name: ResNet
30
31     Description: This class has been created in order to
32                  create, train and evaluate the model of the ResNet.

```

```
31 """
32
33
34 class ResNet:
35     def __init__(self, lr, opt, batch_size, epochs):
36         self.lr = lr
37         self.opt = opt
38         self.batch_size = batch_size
39         self.epochs = epochs
40
41     """
42     Name: trainModel
43
44     Inputs: - model: Keras model that creates the
45             residual network.
46             - X_train: Array that keeps the input data
47             for the training.
48             - y_train: Array that keeps the label of the
49             input data for the training.
50             - X_val: Array that keeps the data of the
51             validation for the training.
52             - y_val: Array that keeps the labels of the
53             validation for the training.
54             - epochs: Integer that says how many epoch
55             has to be trained the model.
56             - batch_size: Integer that keeps the size of
57             the batch for the training.
58             - callbacks: Array that keeps the information
59             for the early stopping and tensorboard's graphics.
60
61     Returns: - history: Array that keeps the model
62             information after being trained.
63
64     Description: This function train the already compiled
65     model and return the results of the training.
66     """
67
68     def trainModel(self, model, X_train, y_train, X_val,
69                   y_val, callbacks):
70         history = model.fit(
71             X_train,
72             y_train,
73             batch_size=self.batch_size,
74             epochs=self.epochs,
75             validation_data=(X_val, y_val),
76             callbacks=callbacks,
77         )
78         return history
```

```

69     """
70     Name: evaluateModel
71
72     Inputs: - model: Model of the residual network.
73             - X_test: Array with the images of the test.
74             - y_test: Array with label of the test.
75
76     Returns: None.
77
78     Description: This function has been created in order
79     to evaluate the already trained model.
80     """
81
82     def evaluateModel(self, model, X_test, y_test):
83         print("-" * 40)
84         print("Evaluating model...")
85         print("-" * 40)
86
87         score = model.evaluate(X_test, y_test, verbose=0)
88
89         print("-" * 40)
90         print("Test loss-> ", score[0])
91         print("Test accuracy-> ", score[1])
92         print("-" * 40)
93
94     """
95     Name: identityBlock
96
97     Inputs: - X: Layers of the model.
98             - f: Size of the convolutional layer's kernel
99             .
100             - filters: Array with the filter's size of
101             the convolutional layers.
102             - block: String that represents the name of
103             the block.
104
105     Returns: - X: Layer of the model.
106
107     Description: This function has been created in order
108     to create an identity block that creates a shortcut that
109     skips one or more layers.
110     """
111
112     def identityBlock(self, X, f, filters, block):
113         # Retrieve filters
114         F1, F2, F3 = filters
115
116         # Create the shortcut
117         shortcutX = X

```



```

112
113     # First component of the main path
114     X = Conv2D(filters=F1, kernel_size=(1, 1), strides
115     =(1, 1), padding="valid")(X)
116     X = BatchNormalization(axis=3)(X)
117     X = Activation("relu")(X)
118
119     # Second component of the main path
120     X = Conv2D(filters=F2, kernel_size=(f, f), strides
121     =(1, 1), padding="same")(X)
122     X = BatchNormalization(axis=3)(X)
123     X = Activation("relu")(X)
124
125     # Third component of the main path
126     X = Conv2D(filters=F3, kernel_size=(1, 1), strides
127     =(1, 1), padding="valid")(X)
128     X = BatchNormalization(axis=3)(X)
129     X = Activation("relu")(X)
130
131     X = layers.Add()([X, shortcutX])
132     x = Activation("relu")(X)
133
134     return X
135
136     """
137     Name: convolutionalBlock
138
139     Inputs: - X: Layers of the model.
140             - f: Size of the convolutional layer's kernel
141             .
142             - filters: Array with the filter's size of
143             the convolotional layers.
144             - block: String that represents the name of
145             the block.
146             - stride: Stride of the layer that combines
147             the main path with the shortcuts.
148
149     Returns: - X: Layer of the model.
150
151     Description: This function has has been created in
152     order to combine the main path of the model with the
153     shortcuts.
154     """
155
156     def convolutionalBlock(self, X, f, filters, block):
157         F1, F2, F3 = filters
158
159         # Create the shortcut
160         shortcutX = X

```

```

152         # Main path
153         X = Conv2D(filters=F1, kernel_size=(1, 1), padding="
154 same")(X)
155         X = BatchNormalization(axis=3)(X)
156         X = Activation("relu")(X)
157
158         # Second component of the main path
159         X = Conv2D(filters=F2, kernel_size=(f, f), padding="
160 same")(X)
161         X = BatchNormalization(axis=3)(X)
162         X = Activation("relu")(X)
163
164         # Third component of the main path
165         X = Conv2D(filters=F3, kernel_size=(1, 1), padding="
166 same")(X)
167         X = BatchNormalization(axis=3)(X)
168
169         # ShortCut path
170         shortcutX = Conv2D(F3, (1, 1))(shortcutX)
171         shortcutX = BatchNormalization(axis=3)(shortcutX)
172
173         # Add shortcut value to the main path and pass it
174         # through a RELU activation
175         X = layers.Add()([X, shortcutX])
176         X = Activation("relu")(X)
177
178         return X
179
180     """
181     Name: buildModel
182
183     Inputs: - lr: Learning rate value in order to build
184 the model.
185             - opt: Type of optimazer in order to build
186 the model.
187
188     Returns: - model: Model already built.
189
190     Description: This function has been created in order
191 to build the residual network model. It returns the model
192 to use it in other functions.
193     """
194
195     def buildModel(self):
196         print("-" * 40)
197         print("Creating model...")
198         print("-" * 40)
199         input_shape = (512, 512, 3)

```

```

193         classes = 2
194
195         X_input = Input(input_shape)
196
197         X = ZeroPadding2D((3, 3))(X_input)
198
199         X = Conv2D(16, (3, 3), padding="same")(X)
200         X = BatchNormalization(axis=3)(X)
201         X = Activation("relu")(X)
202         X = MaxPooling2D((3, 3), padding="same")(X)
203
204         X = self.convolutionalBlock(X, f=3, filters=[32, 32,
128], block="a")
205         X = self.identityBlock(X, 3, [32, 32, 128], block="b"
)
206         X = self.identityBlock(X, 3, [32, 32, 128], block="c"
)
207
208         X = self.convolutionalBlock(X, f=3, filters=[64, 64,
256], block="a")
209         X = self.identityBlock(X, 3, [64, 64, 256], block="b"
)
210         X = self.identityBlock(X, 3, [64, 64, 256], block="c"
)
211         X = self.identityBlock(X, 3, [64, 64, 256], block="d"
)
212
213         X = AveragePooling2D((2, 2))(X)
214
215         X = Conv2D(16, (3, 3), padding="same")(X)
216         X = BatchNormalization(axis=3)(X)
217         X = Activation("relu")(X)
218
219         X = Flatten()(X)
220
221         X = Dense(classes, activation="softmax")(X)
222
223         model = Model(inputs=X_input, outputs=X)
224         if self.opt == 0:
225             model.compile(
226                 optimizer=optimizers.Adam(lr=self.lr),
227                 loss=losses.binary_crossentropy,
228                 metrics=["accuracy"],
229             )
230         elif self.opt == 1:
231             model.compile(
232                 optimizer=optimizers.RMSprop(lr=self.lr),
233                 loss=losses.binary_crossentropy,
234                 metrics=["accuracy"],

```

```

235         )
236         elif self.opt == 2:
237             model.compile(
238                 optimizer=optimizers.SGD(lr=self.lr),
239                 loss=losses.binary_crossentropy,
240                 metrics=["accuracy"],
241             )
242
243         model.summary()
244         return model

```

Listing 3.3: Código de la clase ResNet

## 3.4. Clase AuxFunctions

Esta sección especifica la clase *AuxFunction*, la cual se encargará de almacenar los métodos auxiliares para la creación de la matriz de confusión, del almacenamiento de los pesos de la red y de la creación de la gráfica comparativa de la pérdida entre el conjunto de validación y el conjunto de entrenamiento. Esta especificación puede ser observada en la Tabla 3.4.

### 3.4.1. Especificación de la clase *AuxFunctions*

<b>Nombre de la clase</b>	AuxFunction	
<b>Descripción</b>	Esta clase tiene al funcionalidad de almacenar las funciones auxiliares necesarias para el sistema.	
<b>Datos</b>		
<b>model</b>	Model	Modelo compilado.
<b>history</b>	Dictionary	Es un diccionario que almacena los valores de precisión y pérdida del modelo entrenado.

<b>main_path</b>	String	Es el directorio principal donde se guardarán los resultado de cada una de las funciones de esta clase.
<b>opt</b>	Entero	Es el tipo de optimizador utilizado para la compilación del modelo. Sólo puede tomar los valores 0, 1 o 2.
<b>batch_size</b>	Entero	Es el tamaño del lote que se ha utilizado en cada iteración del entrenamiento.
<b>lr</b>	Decimal	Es el ratio de aprendizaje utilizado por la red para el entrenamiento.
<hr/> <b>Métodos</b>		
<b>create_confusion_matrix</b>	Esta clase es la encargada de crear la matriz de confusión obtenida gracias al previo entrenamiento del modelo. La matriz de confusión será guardada en una imagen en la que tendrá como nombre, el optimizador utilizado, el tamaño del lote utilizado y, por último, el valor del ratio de aprendizaje.	

<b>create_plots_train_test</b>	Esta clase creará una gráfica en la que se mostrará el valor de pérdida tanto para train como para validación después del entrenamiento. La gráfica tendrá como nombre el optimizador utilizado seguido del valor del tamaño del lote y el valor del ratio de aprendizaje.
<b>saveWeights</b>	Esta función se encargará de crear un fichero <i>hdf5</i> cuyo contenido será el valor de los pesos de la red después del entrenamiento. El nombre que tendrá este fichero será el nombre del optimizador utilizado seguido del valor del tamaño del lote y, finalmente, el valor del ratio de aprendizaje.

Tabla 3.4: Especificación de las variables y métodos de la clase AuxFunction

### 3.4.2. Fichero *AuxFunctions.py*

```

1  """
2      Project: Melanoma recognition
3      Author: Juan Jos  M ndez Torrero
4      File: AuxFunctions.py
5      Program: File that contains the additional functions to
        make the network works
6  """
7  import numpy as np
8  import seaborn as sns
9  import matplotlib.pyplot as plt
10 import keras
11
12 from sklearn.metrics import confusion_matrix
13 """
14     Name: AuxFunction
15
16     Description: This class has been created in order to keep
        the additional functions needed to create, train and
        evaluate the ResNet model.
17 """

```

```

18
19 class AuxFunctions:
20     def __init__(self, model, history, main_path, opt,
21 batch_size, lr):
22         self.model = model
23         self.history = history
24         self.main_path = main_path
25         self.opt = opt
26         self.batch_size = batch_size
27         self.lr = lr
28
29     """
30         Name: create_confusion_matrix
31
32         Inputs: - model: ResNet model already trained and
33 compiled.
34                 - X_test: Array that keeps the test data of
35 the model.
36                 - y_test: Array that keeps the test labels of
37 the model.
38
39         Returns: None.
40
41         Description: This function has been created in order
42 to show the confusion matrix of the model. Here we have
43 used an additional library in order to get a confusion
44 matrix image.
45     """
46     def create_confusion_matrix(self, X_test, y_test):
47         print("-" * 40)
48         print("Creating confusion matrix...")
49         print("-" * 40)
50         y_test_confusion_matrix = np.argmax(y_test, axis=1)
51
52         prediction = self.model.predict(X_test)
53         y_pred = np.argmax(prediction, axis=1)
54
55         matrix = confusion_matrix(y_test_confusion_matrix,
56 y_pred)
57         plt.figure(figsize=(10, 10))
58         ax = plt.subplot()
59         sns.heatmap(matrix, annot=True, ax=ax)
60
61         ax.set_xlabel("Predicted labels")
62         ax.set_ylabel("True labels")
63         ax.set_title("Matriz de confusi n")
64         ax.xaxis.set_ticklabels(["malignant", "benign"])
65         ax.yaxis.set_ticklabels(["malignant", "benign"])

```

```

59     plt.savefig(self.main_path + self.opt + "_" + str(
self.batch_size) + "_" + str(self.lr) + "confusion_matrix.
png")
60
61     """
62     Name: create_plots_train_test
63
64     Inputs: - history: Information about the trained
model.
65
66     Returns: None.
67
68     Description: This function creates a graphics with
the train and validation losses. It creates an image with
that values.
69     """
70     def create_plots_train_test(self):
71         print("-" * 40)
72         print("Creating train test plot...")
73         print("-" * 40)
74         plt.clf()
75         plt.plot(self.history.history["loss"])
76         plt.plot(self.history.history["val_loss"], label="
test")
77         plt.xlabel("Train epochs")
78         plt.ylabel("Error")
79         plt.legend(["train", "validation"], loc="lower left")
80         plt.savefig(self.main_path + self.opt + "_" + str(
self.batch_size) + "_" + str(self.lr) + "train-validation.
png")
81
82     """
83     Name: saveWeights
84
85     Inputs: - model: ResNet model.
86             - path: Path where to keep the weights of the
trained model.
87
88     Returns: None.
89
90     Description: The aim of this function is to save the
weights of the model after the training has been completed
.
91     """
92     def saveWeights(self):
93         print("-" * 40)
94         print("Saving weights...")
95         print("-" * 40)

```



```

96         self.model.save_weights(self.main_path + self.opt + "
            _" + str(self.batch_size) + "_" + str(self.lr) + "
            best_weights.hdf5")

```

Listing 3.4: Código de la clase AuxFunctions

### 3.5. Fichero Main

En esta sección se explicará el contenido del fichero *main.py*, el cual es el encargado de unir todas las clases anteriores para poder así realizar el entrenamiento, evaluación y predicción de nuestro modelo de red neuronal residual profunda. Es por esto por lo que esta sección no contará con una especificación sobre este fichero. Se pasa a continuación a mostrar el Listing 3.5 del fichero *main.py*.

```

1  """
2      Project: Melanoma recognition
3      Author: Juan Jos  M ndez Torrero
4      File: main.py
5      Program: Main file to run the project
6  """
7  import os
8
9  # This is to select the graphic card to use
10 os.environ["CUDA_VISIBLE_DEVICES"] = str(1)
11
12 # Python libraries
13 import matplotlib.pyplot as plt
14 import numpy as np
15 import argparse
16 import sys
17 from keras.callbacks import TensorBoard, EarlyStopping
18 from keras.utils import plot_model
19
20 # Additional libraries
21 from ResNet import ResNet
22 from PrepareTrainTest import PrepareTrainTest
23 from AuxFunctions import AuxFunctions
24
25 # Needed inputs in order to run the network
26 parser = argparse.ArgumentParser()
27 parser.add_argument("--epochs", "-e", type=int, help="Number
    of epochs.")
28 parser.add_argument("--learningrate",
29                     "-lr",
30                     type=float,
31                     help="Value of learning rate.")
32 parser.add_argument("--batch",

```

```

33         "-b",
34         type=int,
35         help="Numbers of batch.",
36         default=10)
37 parser.add_argument("--datasets", "-d", type=str, help="Path
    for h5py files.")
38 parser.add_argument("--traintest",
39                     "-tt",
40                     type=str,
41                     help="Name of the train test validation
        hdf5 file.")
42 parser.add_argument(
43     "--optimizer",
44     "-o",
45     type=int,
46     help="Optimizer used: 0.Adam 1.RMSprop 2.SGD",
47     default=0,
48 )
49 parser.add_argument(
50     "--patience",
51     "-p",
52     type=int,
53     help="Number of epochs without improving validation loss.
        ",
54 )
55 args = parser.parse_args()
56
57 epochs = args.epochs
58 lr = args.learningrate
59 batch_size = args.batch
60 path_datasets = args.datasets
61 file_name = args.traintest
62 opt = args.optimizer
63 patience = args.patience
64
65 if epochs <= 0:
66     print("Error! The number of epochs must be greater than 0
        ")
67     print("Please, run main.py -h to see the options")
68     sys.exit(-1)
69 elif batch_size <= 0:
70     print("Error! Batch size must be greater than 0")
71     print("Please, run main.py -h to see the options")
72     sys.exit(-1)
73 elif opt != 0 and opt != 1 and opt != 2:
74     print("Error! The optimizer must be 0, 1 or 2")
75     print("Please, run main.py -h to see the options")
76     sys.exit(-1)
77 elif patience <= 0:

```

```
78     print("Error! Patience must be greater than 0")
79     print("Please, run main.py -h to see the options")
80     sys.exit(-1)
81
82 # Prepare train-test data section
83
84 ptt = PrepareTrainTest(path_datasets, file_name)
85
86 if os.path.isdir(path_datasets):
87     # PrepareTrainTest.createTrainTestH5PY(
88     malignant_equalized, benign_equalized)
89     X_train, X_test, X_val, y_train, y_test, y_val = ptt.
90     readDataH5PY()
91 else:
92     print("Error! H5PY file does not exists...")
93     sys.exit(-1)
94
95 rn = ResNet(lr, opt, batch_size, epochs)
96 # Start the section where the model is built
97 model = rn.buildModel()
98
99 if opt == 0:
100     type_opt = "Adam"
101 elif opt == 1:
102     type_opt = "RMSprop"
103 elif opt == 2:
104     type_opt = "SGD"
105
106 # Creation of the model's callbacks Tensorboard to create the
107 # graphs with the results and EarlyStopping to stop the
108 # training when it is not improving
109 callbacks = [
110     TensorBoard(
111         log_dir="./logs/" + type_opt + "_b" + str(batch_size)
112         + "_" + str(lr) + "/",
113         write_images=True,
114         write_graph=True,
115         update_freq="epoch",
116     ),
117     EarlyStopping(
118         monitor="val_loss",
119         mode="min",
120         patience=patience,
121         verbose=1,
122         restore_best_weights=True,
123     ),
124 ]
```

```
121 history = rn.trainModel(model, X_train, y_train, X_val, y_val
    , callbacks)
122
123 print("-" * 40)
124 print("Summary:")
125 print("\t optimizer -> " + type_opt)
126 print("\t lr -> ", lr)
127 print("\t epochs -> ", epochs)
128 print("\t batch size -> ", batch_size)
129 print("\t patience -> ", patience)
130 print("-" * 40)
131
132 rn.evaluateModel(model, X_test, y_test)
133
134 if os.path.exists("./results/") == False:
135     print("Creating results directory...")
136     os.mkdir("./results/")
137
138 ax = AuxFunctions(model, history, "results/", type_opt,
    batch_size, lr)
139 ax.create_confusion_matrix(X_test, y_test)
140 ax.create_plots_train_test()
141 ax.saveWeights()
```

Listing 3.5: Código del fichero Main