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
D. Javier Sánchez Monedero

Septiembre, 2019

UNIVERSIDAD D CÓRDOBA

Índice general

Listings

Índice de tablas

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 *malignas* 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 para la creación de este proyecto, 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 la creación de 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 se 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 las que se encuentran las imágenes para su clasificación inicial.

main_path	String	Identifica la ruta en donde se guardarán los resultados de esta clasificación.
Métodos		
createHD5PY	Guarda las imágenes encontradas en el directorio *path_images* en el directorio *main_path*. El formato de la salida es *hdf5*, en el cual se encontrarán tanto las imágenes encontradas divididas según el tipo de lesión.	
readDataH5PY	Lee las imágenes guardadas en el fichero *hdf5* del directorio *main_path*. 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*

```
"""
    Project: Melanoma recognition
    Author: Juan Jos  M ndez Torrero
    File: PrepareData.py
    Program: File to prepare the images and save them onto a
   h5py file
"""
import numpy as np
import h5py
import os
import json

from tqdm import tqdm  # This library is used to know the
   percentage of images already classified and saved
"""
    Name: PrepareData

```

```
16      Description: This class has been created in order to
    classify the images before the training.
17  """
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
    save the images.
30
31          Returns: None.
32
33          Description: This function classifies the images into
     malignant and benign. After that, creates a hdf5 file in
    order to save them.
34      """
35      def createH5PY(self):
36          files = []
37          with h5py.File(self.path_datasets + "dataset.hdf5", "
    w") as hdf:
38              benign_group = hdf.create_group("benign_images")
39              malignant_group = hdf.create_group("
    malignant_images")
40              print("Creating h5py file...")
41              # Read all images from directory
42              for r, d, f in os.walk(self.path_images):
43                  for file in tqdm(f):
44                      if ".json" in file:
45                          paths = os.path.join(r, file)
46                          with open(paths) as json_file:
47                              data_json = json.load(json_file)
48                              if (data_json["meta"]["clinical"]
49                                  ["benign_malignant"] == "
    malignant"):
50                                  data = image.load_img(paths
    [:-4] + "jpg",
51
    target_size=[512, 512])
52                                  malignant_group.
    create_dataset(
53                                      file[:-4] + "jpg",
54                                      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
92              benign_items = list(hdf.get("benign_images").
    items())
93              for items in tqdm(benign_items):
94                  benign_images.append(
95                      np.array(hdf.get("benign_images").get(
    items[0])))
96
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 las imáegenes 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 (*train)*, una para los test (*test*) y otra para la validación (*validation*). 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

file_name	String	Nombre del fichero en donde serán guardadas las imágenes con sus etiquetas.
Métodos		
createTrainTestH5PY	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 *main_path* con el nombre *file_name*.	
readDataH5PY	Esta función se encarga de leer el fichero *file_name* 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*

```
"""
    Project: Melanoma recognition
    Author: Juan Jos  M ndez Torrero
    File: PrepareTrainTest.py
    Program: File that creates the train test file in order
   to train the model with
"""

import h5py
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import numpy as np
from keras.utils import np_utils
"""
    Name: PrepareTrainTest
```

```
15
16      Description: This class has been created in order split
        the input data into train, test and validation for the
        training.
17  """
18
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
        file that keeps the train, test and validation data.
30                  - malignant_images: Array with the malignant
        images.
31                  - benign_images: Array with the benign images
        .
32
33          Returns: None.
34
35          Description: This function splits the recieved data
        into train, test and validation. At the end, it saves the
        data into a hdf5 file.
36      """
37      def createTrainTestH5PY(self, malignant_images,
        benign_images):
38          print("-" * 40)
39          print("Splitting data into train, test and validation
        ...")
40          print("-" * 40)
41          # Concatenate all data and create labels
42          all_data = np.vstack((malignant_images, benign_images
        ))
43          labels = np.concatenate(
44              (np.zeros(len(malignant_images)), np.ones(len(
        benign_images))))
45
46          # Do train test split in order to get the training,
        test and validation data
47          X_train, X_test, y_train, y_test = train_test_split(
        all_data,
48
        labels,
```

```
49
   test_size=0.4,
50
   stratify=labels)
51      X_val, X_test, y_val, y_test = train_test_split(
   X_test,
52
   y_test,
53
   test_size=0.5,
54
   stratify=y_test)
55
56      # Make labels categorical
57      y_train = np_utils.to_categorical(y_train)
58      y_test = np_utils.to_categorical(y_test)
59      y_val = np_utils.to_categorical(y_val)
60
61      print("-" * 40)
62      print("Creating h5py train test file...")
63      print("-" * 40)
64      dataset_output_path = self.main_path + self.file_name
65      with h5py.File(dataset_output_path, "w") as hdf:
66          hdf.create_dataset("X_train", data=X_train,
   compression="gzip")
67          hdf.create_dataset("y_train", data=y_train,
   compression="gzip")
68          hdf.create_dataset("X_test", data=X_test,
   compression="gzip")
69          hdf.create_dataset("y_test", data=y_test,
   compression="gzip")
70          hdf.create_dataset("X_val", data=X_val,
   compression="gzip")
71          hdf.create_dataset("y_val", data=y_val,
   compression="gzip")
72
73    """
74      Name: readDataH5PY
75
76      Inputs: - path: Path of the hdf5 file.
77              - name_file: String for the name of the hdf5
   file that keeps the train, test and validation data.
78
79      Returns: - dataset: A dictionary that keeps the value
    of the train, test and validation data.
80
81      Description: This function reads the data for the
   training from a hdf5 file and returns a dictionary that
   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,
    "r")
88          return (
89              dataset["X_train"][()],
90              dataset["X_test"][()],
91              dataset["X_val"][()],
92              dataset["y_train"][()],
93              dataset["y_test"][()],
94              dataset["y_val"][()],
95          )
```

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.

opt	Entero	Identifica el tipo de optimizador que la red utilizará para compilar el modelo. Su valor será 0, 1 o 2.
batch_size	Entero	Identifica el tamaño del lote que se entrenará cada iteración.
epochs	Entero	Identifica el máximo número de iteraciones que nuestro modelo entrenará.
Métodos		
trainModel	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.	
evaluateModel	Esta función consiste en la evaluación de nuestro modelo una vez entrenado. Esta función nos mostrará por pantalla el valor de precisión y perdida según los datos utilizados para el test.	
identityBlock	Esta función ha sido creada para simular el atajo hablado en el Capítulo **??**.	
convolutionalBlock	Esta función ha sido creada para simular unir el atajo creado por la anterior función y el flujo principal de la red.	

buildModel	Esta función es la encargada de unir todo el flujo de capas de nuetra 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*

```
"""
    Project: Melanoma recognition
    Author: Juan Jos  M ndez Torrero
    File: ResNet.py
    Program: File that creates the model and trains it with
   the images already processed
"""
import tensorflow as tf
import numpy as np

import keras
from keras.layers import (
    Dense,
    Activation,
    Flatten,
    Conv2D,
    MaxPooling2D,
    BatchNormalization,
    ZeroPadding2D,
    Input,
    AveragePooling2D,
)
from keras import Model
from keras import optimizers
from keras import losses
from keras import layers

"""
    Name: ResNet

    Description: This class has been created in order to
   create, train and evaluate the model of the ResNet.
```

```
"""


class ResNet:
    def __init__(self, lr, opt, batch_size, epochs):
        self.lr = lr
        self.opt = opt
        self.batch_size = batch_size
        self.epochs = epochs

    """
        Name: trainModel

        Inputs: - model: Keras model that creates the residual network.
                - X_train: Array that keeps the input data for the training.
                - y_train: Array that keeps the label of the input data for the training.
                - X_val: Array that keeps the data of the validation for the training.
                - y_val: Array that keeps the labels of the validation for the training.
                - epochs: Integer that says how many epoch has to be trained the model.
                - batch_size: Integer that keeps the size of the batch for the training.
                - callbacks: Array that keeps the information for the early stopping and tensorboard's graphics.

        Returns: - history: Array that keeps the model information after being trained.

        Description: This function train the already compiled model and return the results of the training.
    """

    def trainModel(self, model, X_train, y_train, X_val, y_val, callbacks):
        history = model.fit(
            X_train,
            y_train,
            batch_size=self.batch_size,
            epochs=self.epochs,
            validation_data=(X_val, y_val),
            callbacks=callbacks,
        )
        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
    to evaluate the already trained model.
79      """
80
81      def evaluateModel(self, model, X_test, y_test):
82          print("-" * 40)
83          print("Evaluating model...")
84          print("-" * 40)
85
86          score = model.evaluate(X_test, y_test, verbose=0)
87
88          print("-" * 40)
89          print("Test loss-> ", score[0])
90          print("Test accuracy-> ", score[1])
91          print("-" * 40)
92
93      """
94          Name: identityBlock
95
96          Inputs: - X: Layers of the model.
97                  - f: Size of the convolutional layer's kernel
    .
98                  - filters: Array with the filter's size of
    the convolotional layers.
99                  - block: String that represents the name of
    the block.
100
101         Returns: - X: Layer of the model.
102
103         Description: This function has been created in order
    to create an identity block that creates a shortcut that
    skips one or more layers.
104     """
105
106     def identityBlock(self, X, f, filters, block):
107         # Retrieve filters
108         F1, F2, F3 = filters
109
110         # Create the shortcut
111         shortcutX = X
```

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

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

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

main_path	String	Es el directorio principal donde se guardarán los resultado de cada una de las funciones de esta clase.
opt	Entero	Es el tipo de optimizador utilizado para la compilación del modelo. Sólo puede tomar los valores 0, 1 o 2.
batch_size	Entero	Es el tamaño del lote que se ha utilizado en cada iteración del entrenamiento.
lr	Decimal	Es el ratio de aprendizaje utilizado por la red para el entrenamiento.
Métodos		
create_confusion_matrix	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.	

create_plots_train_test	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.
saveWeights	Esta función se encargará de crear un fichero *hdf5* 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*

```
"""
    Project: Melanoma recognition
    Author: Juan Jos  M ndez Torrero
    File: AuxFunctions.py
    Program: File that contains the additional functions to
   make the network works
"""
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import keras

from sklearn.metrics import confusion_matrix
"""
    Name: AuxFunction

    Description: This class has been created in order to keep
    the additional functions needed to create, train and
   evaluate the ResNet model.
"""
```

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

```
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*.

```
"""
    Project: Melanoma recognition
    Author: Juan Jos  M ndez Torrero
    File: main.py
    Program: Main file to run the project
"""
import os

# This is to select the graphic card to use
os.environ["CUDA_VISIBLE_DEVICES"] = str(1)

# Python libraries
import matplotlib.pyplot as plt
import numpy as np
import argparse
import sys
from keras.callbacks import TensorBoard, EarlyStopping
from keras.utils import plot_model

# Additional libraries
from ResNet import ResNet
from PrepareTrainTest import PrepareTrainTest
from AuxFunctions import AuxFunctions

# Needed inputs in order to run the network
parser = argparse.ArgumentParser()
parser.add_argument("--epochs", "-e", type=int, help="Number
    of epochs.")
parser.add_argument("--learningrate",
                    "-lr",
                    type=float,
                    help="Value of learning rate.")
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(
        malignant_equalized, benign_equalized)
88      X_train, X_test, X_val, y_train, y_test, y_val = ptt.
        readDataH5PY()
89  else:
90      print("Error! H5PY file does not exists...")
91      sys.exit(-1)
92
93  rn = ResNet(lr, opt, batch_size, epochs)
94  # Start the section where the model is built
95  model = rn.buildModel()
96
97  if opt == 0:
98      type_opt = "Adam"
99  elif opt == 1:
100     type_opt = "RMSprop"
101 elif opt == 2:
102     type_opt = "SGD"
103
104 # Creation of the model's callbacks Tensorboard to create the
        graphs with the results and EarlyStopping to stop the
        training when it is not improving
105 callbacks = [
106     TensorBoard(
107         log_dir="./logs/" + type_opt + "_b" + str(batch_size)
        + "_" + str(lr) + "/",
108         write_images=True,
109         write_graph=True,
110         update_freq="epoch",
111     ),
112     EarlyStopping(
113         monitor="val_loss",
114         mode="min",
115         patience=patience,
116         verbose=1,
117         restore_best_weights=True,
118     ),
119 ]
120
```

```
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