Universidad de Granada
Escuela técnica Superior de ingeniería
informática y telecomunicaciones

Máster
Ingeniería Informática
Segundo cuatrimestre

Sistemas Inteligentes para la Gestión de Empresa.

Trabajo de Teoría - Clasificación de imágenes médicas

Ángel Murcia Díaz
DNI: 26821637A
angelmd96@correo.ugr.es

Curso académico 2019-2020
Granada, 25 de enero de 2021

Índice

Índice de figuras

1. Introducción al tema y motivación

El tema seleccionado para el trabajo es el llamado "Clasificación de imágenes médicas", concretamente el trabajo se centra en *detección del COVID-19 en imágenes médicas utilizando técnicas de Deep Learning*, para ello se utiliza como referencia el siguiente artículo web [1], el cuál a parte de información acerca del problema tratado, cuenta con soluciones ya implementadas para resolverlo.

Posteriormente se detalla más en profundidad el problema, el conjunto de datos de imágenes, así como se especifica las herramientas que se utilizan en el articulo web [1] para resolverlo y que se utilizan en este trabajo para comprobar y explicar su funcionamiento, así como para intentar mejorar el resultado realizando una serie de modificaciones.

Las razones que me han llevado a seleccionar este tema, es decir, lo que me ha motivarlo a elegirlo ha sido:

- El *Deep Learning* siempre ha sido un tema que me ha llamado la atención y que me ha gustado, también he trabajado con este tema en el pasado utilizando múltiples lenguajes de programación y tecnologías diferentes.
- El problema a resolver está a la orden del día, puesto que es "lo que está viviendo todo el mundo", también me llamó la atención puesto que sé que actualmente es un tema que se está intentando resolver también por profesionales en el campo.
- Llevar esto a la práctica real puede ser muy útil, puesto que es un análisis automatizado para ahorrar a los profesionales médicos un tiempo valioso.

2. Herramientas y tecnologías utilizadas

Las tecnologías y herramientas que se utilizan en el articulo web [1] y que por lo tanto se utilizan en este trabajo son:

- *Python* como lenguaje de programación.
- La librería *Keras* sobre *Tensorflow* como herramienta para la construcción de modelo de *Deep Learning*, puesto que de forma muy sencilla y con pocas líneas de código se pueden conseguir modelos de *Deep Learning* muy complejos, esta librería es muy conocida y utilizada por ser amigable para el usuario, modular y extensible

Destacar que como elección personal, se utilizará *Google Colab* [2], como entorno de ejecución de *Python* desde *Google*, puesto que se puede utilizar desde cualquier navegador web y sobretodo porque permite activar la "GPU", muy importante a la

hora de computar los modelos que se van a crear en términos de velocidad. Destacar que el cambio que hay que hacer para ejecutar el script en *Google colab* [2] es muy breve, tan solo tengo que conectar mi **Drive** [3] (con el conjunto de imágenes subido) a mi cuaderno de *Google colab* y eliminar los argumentos en consola transformándolos en variables normales.

Figura 1: Logos de tecnologías y herramientas

3. Análisis del problema

A continuación se analizará en profundidad el problema.

El problema se trata de clasificar un conjunto de imágenes de rayos X de los pulmones de un conjunto de pacientes, el objetivo del problema es saber si el paciente al que pertenece la imagen tiene el COVID-19 (Coronavirus) o no lo tiene. Por lo tanto consideramos un problema de clasificación binaria de imágenes, las dos posibles categorías a las que puede pertenecer una imagen son:

- Tiene COVID-19.
- No tiene COVID-19, es "normal".

Para esto se implementarán modelos de *Deep Learning*, utilizando las herramientas y ténicas nombradas en la sección anterior del documento, concretamente se recurrirá al uso de *redes neuronales profundas (CNN)*, que funcionan bastante bien en el campo de la clasificación de imágenes.

Figura 2: Imagen del articulo web [1]

4. Análisis del conjunto de datos

En esta sección se va a realizar un análisis del conjunto de datos que se utiliza, es decir, de las imágenes de los pulmones de pacientes, destacar que son imágenes de rayos X de pacientes reales.

Las imágenes se representan a través del conjunto de píxeles, los valores de los píxeles toman un valor entre 0 y 255. Utilizando los datos de los píxeles de una imagen se trabaja, de tal forma que por cada imagen, se introduce el valor de todos sus píxeles en la capa de entrada de la *red neuronal profunda* que se utiliza para resolver el problema. Destacar que la librería **OpenCV**[4] es la encargada de realizar esta labor de carga de imágenes, haciéndola totalmente trasparente al usuario.

A continuación en la figura 3, se muestran las imágenes de cada clase, para que el lector pueda hacerse una idea de las características que pueden tener las imágenes de pulmones que representan enfermos de COVID-19 y las características que pueden tener las imágenes de pulmones que representan personas sanas.

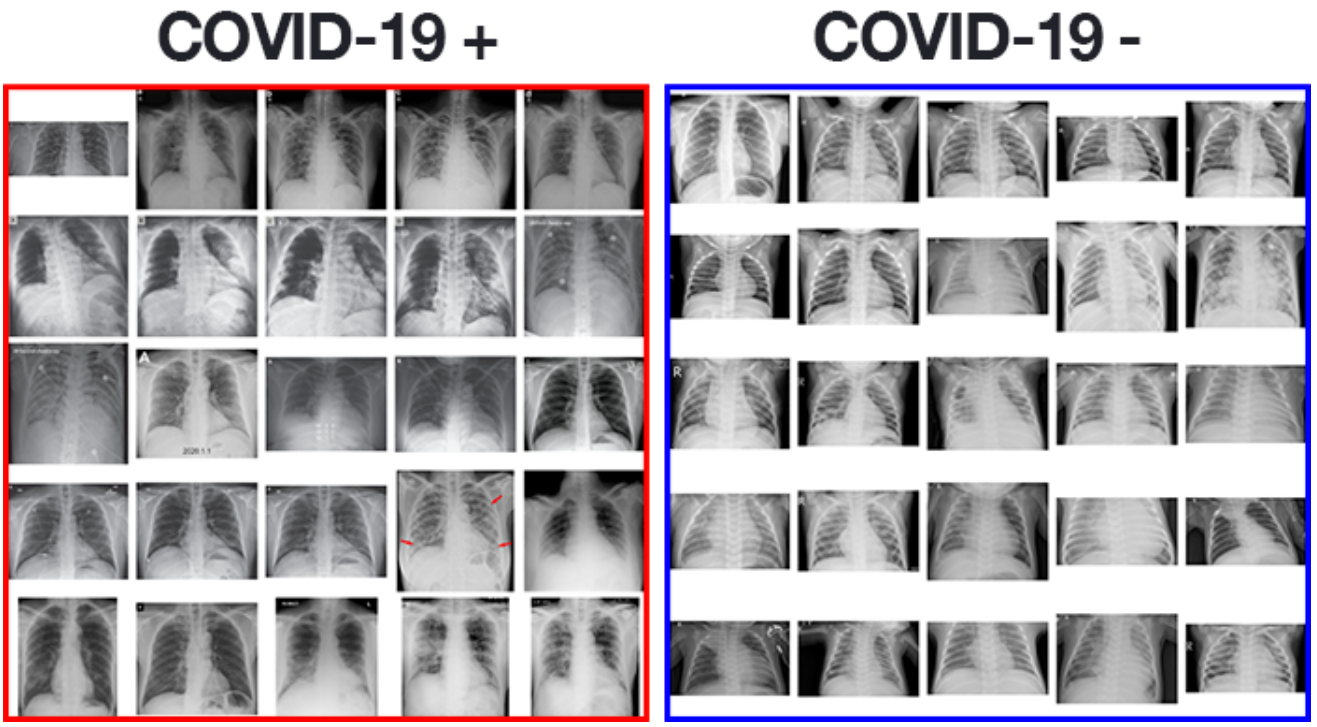

Figura 3: Ejemplos de imágenes de ambas categorías

El conjunto de datos de imágenes de rayos X COVID-19 fue recopilado por el Dr. Joseph Cohen, becario postdoctoral en la Universidad de Montreal.

El Dr. Cohen comenzó a recopilar imágenes de rayos X de los casos de COVID-19 hace unas semana y a publicarlas en un repositorio de GitHub [5]. En este repositorio también hay imágenes de pacientes que padecen MERS, SARS, y ARDS, por lo que solo se toman las imágenes de los enfermos de COVID-19.

El conjunto de imágenes de rayos X de pacientes sanos se tomo de un conjunto de datos de Kaggle (Chest X-Ray Images (Pneumonia)) [6], de donde se toman las muestras de imágenes de rayos X de pacientes sanos.

5. Implementación del Script Original de *Deep Learning* utilizando Keras

Durante esta sección del trabajo se va a explicar el script original publicado en el artículo web [1].

5.1. Puesta a punto

En primer lugar, se tiene que descargar el archivo "keras-covid-19" en formato *zip*, que se puede solicitar a través de correo electrónico en la misma página web

donde está el articulo [1], este contiene:

- Los conjuntos de datos de imágenes de rayos X.
- El script con la aplicación de *Deep Learning*.
- Otros archivos que no se utilizarán, como el script de extracción del conjunto de imágenes.

En segundo lugar, es necesario descomprimir el archivo y trabajar sobre ese mismo script, puesto que si queremos ejecutar el script directamente con el comando que muestra el tutorial, tiene estrictamente que seguir esta estructura de carpetas.

5.2. Carga de librerías

La primera parte del script corresponde a la carga o importación de las librerías de *Python* necesarias para que se ejecute sin problemas, destacar que es necesario instalarlas utilizando "pip" en caso de que no se encuentren instaladas, esta parte del código se puede ver a continuación:

```
# USAGE
# python train_COVID-19.py --dataset dataset

# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import cv2
import os
```

5.3. Construcción del analizador de argumentos

Esta parte de código consiste en construir un analizador de argumentos pasados por consola, de tal forma que cuando se ejecute el programa a continuación en la consola se puedan poner los argumentos que se deseen, utilizando banderas para distinguir cada argumento.

```
26 # construct the argument parser and parse the arguments
27 ap = argparse.ArgumentParser()
28 ap.add_argument("-d", "--dataset", required=True,
29    help="path to input dataset")
30 ap.add_argument("-p", "--plot", type=str, default="plot.png",
31    help="path to output loss/accuracy plot")
32 ap.add_argument("-m", "--model", type=str, default="COVID-19.model",
33    help="path to output loss/accuracy plot")
34 args = vars(ap.parse_args())
```

Los argumentos que se le pueden pasar al programa por consola son:

- –dataset o -d: es la ruta de los conjuntos de datos a utilizar. Es necesario especificarlo para que el programa pueda ejecutarse.
- –plot o -p: es la ruta del fichero de plot que se forma durante la ejecución del programa. No es necesario especificarlo puesto que tiene un argumento por defecto.
- –model o -m: es la ruta del fichero que guardará el modelo generado para poder utilizarlo posteriormente si el usuario lo desea. No es necesario especificarlo puesto que tiene un argumento por defecto.

Un ejemplo de ejecución del programa utilizando todos los argumentos es:

```
python3 train_COVID-19.py --dataset dataset --plot plot.jpg --model
    modelo-COVID-19
```

Otros parámetros interesantes como son:

- Factor de aprendizaje: es el parámetro que modera las actualizaciones de los pesos, de tal forma que si tiene un valor alto en cada actualización de pesos "aprenderá" bastante o se adaptará bastante a los datos de entrenamiento, mientras que si es bajo "aprenderá" poco.
- Numero de épocas: es el número de iteraciones que durará el proceso de aprendizaje, destacar que no por más épocas tendremos mejores resultados ya que podemos caer en el llamado sobreaprendizaje.

- Tamaño del lote de aprendizaje (batch): es cada cuantas imágenes aprende el algoritmo, si tenemos un tamaño de lote de 10, la red neuronal no modificará los pesos hasta que pasen 10 imágenes de entrenamiento.

Se definen directamente como variables en el código del script.

```
36 # initialize the initial learning rate, number of epochs to train for,
37 # and batch size
38 INIT_LR = 1e-3
39 EPOCHS = 2
40 BS = 8
```

5.4. Cargado y preprocesado de los datos

A continuación, se procede a cargar las imágenes que se utilizarán para este experimento, para eso se explora en la ruta que se le proporciona como argumento (dataset) y se consiguen todas las rutas completas de cada uno de los archivos de imagen, que se guardan en una lista.

Posteriormente recorriendo la lista, se extrae la clase a la que pertenece cada imagen, puesto que las imágenes vienen divididas en dos carpetas (COVID-19 y normal). También se carga cada una de las imágenes utilizando funciones de la librería *OpenCV* [4].

Tanto la imagen como la clase de cada una de las imágenes se va agregando a vectores, uno contiene todas las imágenes cargadas y el otro la etiqueta de cada una de las imágenes, es decir, a que clase pertenece cada imagen.

```
42 # grab the list of images in our dataset directory, then initialize
43 # the list of data (i.e., images) and class images
44 print("[INFO] loading images...")
45 imagePaths = list(paths.list_images(args["dataset"]))
46 data = []
47 labels = []
48
49 # loop over the image paths
50 for imagePath in imagePaths:
51    # extract the class label from the filename
52    label = imagePath.split(os.path.sep)[-2]
53
54    # load the image, swap color channels, and resize it to be a fixed
55    # 224x224 pixels while ignoring aspect ratio
56    image = cv2.imread(imagePath)
57    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
58    image = cv2.resize(image, (224, 224))
59
60    # update the data and labels lists, respectively
```

```
61    data.append(image)
62    labels.append(label)
```

Posteriormente se le aplica un preprocesado muy básico, tanto al vector que contiene las imágenes como al vector que contiene las etiquetas.

El vector de imágenes se normaliza, de tal forma que los valores de los píxeles de las imágenes pasan de estar en el intervalo [0, 255] al intervalo [0, 1], esto es porque cuando se trabaja con *redes neuronales profundas* siempre es recomendable normalizar los datos, aunque como en este caso se sepa de antemano que todos los datos están en el mismo intervalo original.

El vector de etiquetas se binariza, para que en lugar de que una etiqueta sea igual a "COVID-19" o "normal", la etiqueta sea un vector de tamaño 2 donde la primera posición corresponda a "COVID-19" y la segunda a normal y que se codifique con un 1 para la clase a la que pertenece y un 0 para la clase a la que no pertenece. Siempre se debería hacer este paso cuando se trabaja con un problema de clasificación, sobretodo si es multiclase aunque en este caso sea un problema de clasificación binaria.

```
64 # convert the data and labels to NumPy arrays while scaling the pixel
65 # intensities to the range [0, 255]
66 data = np.array(data) / 255.0
67 labels = np.array(labels)
68
69 # perform one-hot encoding on the labels
70 lb = LabelBinarizer()
71 labels = lb.fit_transform(labels)
72 labels = to_categorical(labels)
```

Posteriormente se pasa a separar el conjunto de datos en entrenamiento y prueba, de esta manera tenemos una parte para realizar el entrenamiento de la *red neuronal profunda* y la otra parte para validar la *red neuronal profunda*, es decir, ver lo bueno que es el modelo construido y entrenado. Destacar que esta partición se realiza de forma estratificada lo que quiere decir que se respeta la proporción de las clases y que tendremos un problema balanceado puesto que tendremos los mismos ejemplos de ambas clases tanto en el conjunto de entrenamiento como en el conjunto de prueba. El fragmento de código que hace este cometido es:

```
74 # partition the data into training and testing splits using 80% of
75 # the data for training and the remaining 20% for testing
76 (trainX, testX, trainY, testY) = train_test_split(data, labels,
77    test_size=0.20, stratify=labels, random_state=42)
```

Por último, debido a que tenemos muy pocas imágenes para entrenar, se va a utilizar un generador, esto lo que hace es generar nuevas imágenes a partir de las existentes con operaciones como por ejemplo rotación.

```
79 # initialize the training data augmentation object
80 trainAug = ImageDataGenerator(
81    rotation_range=15,
82    fill_mode="nearest")
```

En este caso en concreto se están generando nuevas imágenes variando el rango de rotación y utilizando la filosofía del vecino más cercano en los puntos de fuera de los límites.

5.5. Construcción y compilación del modelo

Ahora se procede a la construcción del modelo, es decir, a la construcción de la *red neuronal convolucional* (CNN).

Para ello, se utilizará una red de base a la cuál le añadiremos una serie de capas que se le adjuntarán al final de la red base.

La red de base que se utiliza es "VGG16", esta es una red entrenada con pesos pre-entrenados en ImageNet, dejando fuera la parte "top" (será la parte que se añadirá manualmente), por último destacar que las imágenes se introducen en la red como tensor.

La arquitectura de la red "VGG16" es que se puede ver en la figura 4.

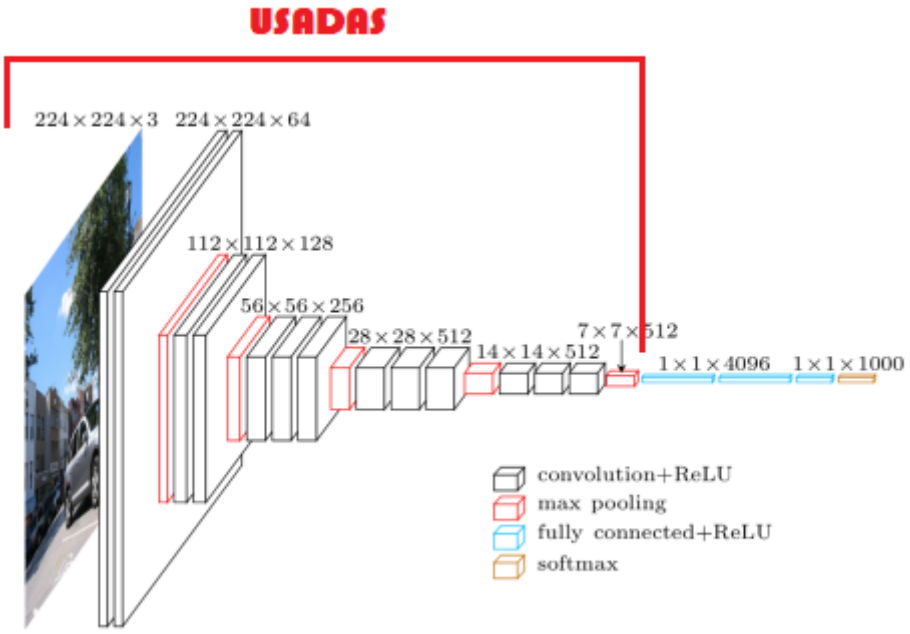

Figura 4: Red básica "VGG16"

De la cuál se utilizarán las capas señaladas en la imagen, el resto se sustituirán por las siguientes capas:

- Una capa de polling con un filtro de 4 x 4, para reducir la complejidad de las fotografías, así como para concentrar la información.
- Una capa Flatten, es una capa de aplanado, convierte los elementos de la matriz de imágenes de entrada en un array plano.
- Una capa densa de 64 neuronas con "Relu" como función de activación.
- Una capa de Dropout, que nos permite regular cuál es la probabilidad de mantener los valores de entrada de dicha neurona en tiempo de entrenamiento. Si por ejemplo asignamos un DropOut entre la capa 2 y 3 de nuestro modelo, y decimos que el parámetro P es de 0.4 estamos diciendo que todos los valores que van a salir de la capa 2 en tiempo de entrenamiento tienen un 40 % de probabilidad de llegar a la capa 3, todos los demás serán forzados a ser 0.
- Una capa de salida con 2 neuronas con función de activación "Softmax", puesto que contamos con 2 clases (problema de clasificación binaria).

Por último, se le dice a la red que congele los pesos correspondientes a la red base, por lo que solo cambiarán los pesos de la parte que se ha definido.

```
84  # load the VGG16 network, ensuring the head FC layer sets are left
85  # off
86  baseModel = VGG16(weights="imagenet", include_top=False,
87     input_tensor=Input(shape=(224, 224, 3)))
88
89  # construct the head of the model that will be placed on top of the
90  # the base model
91  headModel = baseModel.output
92  headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
93  headModel = Flatten(name="flatten")(headModel)
94  headModel = Dense(64, activation="relu")(headModel)
95  headModel = Dropout(0.5)(headModel)
96  headModel = Dense(2, activation="softmax")(headModel)
97
98  # place the head FC model on top of the base model (this will become
99  # the actual model we will train)
100 model = Model(inputs=baseModel.input, outputs=headModel)
101
102 # loop over all layers in the base model and freeze them so they will
103 # *not* be updated during the first training process
104 for layer in baseModel.layers:
105    layer.trainable = False
```

A continuación, se procede con el compilado del modelo, para esto hay que especificar el optimizador, la función de pérdida y la métrica.

```
# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
   metrics=["accuracy"])
```

- El optimizador utilizado es "Adam" al cuál se le introducen como parámetros el factor de aprendizaje (especificado al inicio del script), así como el decremento de ese factor de aprendizaje con el desarrollo de las épocas el cuál es el propio factor de aprendizaje entre el número de épocas.
- La función de pérdida es la entropía cruzada binaria, la cuál funciona bien de forma teórica para problemas de clasificación binaria.
- Por último, la métrica especificada es el *accuracy*.

5.6. Entrenamiento del modelo

A continuación, se entrena la *red neuronal convolucional*, utilizando el modelo que se ha creado anteriormente.

```
# train the head of the network
print("[INFO] training head...")
H = model.fit_generator(
   trainAug.flow(trainX, trainY, batch_size=BS),
   steps_per_epoch=len(trainX) // BS,
   validation_data=(testX, testY),
   validation_steps=len(testX) // BS,
   epochs=EPOCHS)
```

También se define el conjunto de prueba como conjunto de validación para que en el entrenamiento se pueda ir comprobando el progreso conforme va entrenando la red, el número de épocas, y número de steps por época, este último parámetro dependerá tanto del tamaño del conjunto de imágenes de entrenamiento como del tamaño del lote que se hubiera especificado anteriormente.

5.7. Predicción sobre del modelo

A continuación, comprobará la bondad de la *red neuronal convolucional*.

```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

```

```
126 # for each image in the testing set we need to find the index of the
127 # label with corresponding largest predicted probability
128 predIdxs = np.argmax(predIdxs, axis=1)
```

Para esto pasamos todas las imágenes de prueba y contrastamos la predicción de la red con la clase original a la que pertenece cada imagen.

5.8. Impresión de resultados

A continuación en el script se diseña una digna impresión de resultados donde aparecen las siguientes métricas:

- Matriz de confusión: es una matriz que enfrenta los valores reales que tiene el conjunto de datos frente a los predichos por el modelo predictivo. De tal forma que a simple vista se pueda saber los aciertos y los errores del modelo predictivo en cada una de las clases.
- Accuracy: es el porcentaje de patrones bien clasificados, es una buena medida de bondad puesto que mide el porcentaje de acierto del modelo, aunque puede ser engañoso en algunos casos, como cuando tenemos un problema desbalanceado.
- Sensitivity: es el porcentaje de verdaderos positivos correctos.
- Specificity: es el porcentaje de verdaderos negativos correctos.

```
134 # compute the confusion matrix and and use it to derive the raw
135 # accuracy, sensitivity, and specificity
136 cm = confusion_matrix(testY.argmax(axis=1), predIdxs)
137 total = sum(sum(cm))
138 acc = (cm[0, 0] + cm[1, 1]) / total
139 sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
140 specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
141
142 # show the confusion matrix, accuracy, sensitivity, and specificity
143 print(cm)
144 print("acc: {:.4f}".format(acc))
145 print("sensitivity: {:.4f}".format(sensitivity))
146 print("specificity: {:.4f}".format(specificity))
```

5.9. Impresión de gráficos

Para ir comprobando la evolución tanto del conjunto de entrenamiento y como del conjunto de validación se muestra en una gráfica la evolución del *accuracy* y del

loss (que es el error que comete la *red neuronal convolucional*), de tal forma que si la red está siendo entrenada correctamente el error va decrementando conforme pasan las épocas mientras que el porcentaje de patrones bien clasificados sube.

Destacar que al utilizarse el conjunto de prueba como conjunto de validación, se puede ir comprobando como va mejorando o empeorando la clasificación de las imágenes sobre el conjunto de prueba final conforme transcurren las iteraciones.

El código del script original perteneciente a esta parte es:

```
148 # plot the training loss and accuracy
149 N = EPOCHS
150 plt.style.use("ggplot")
151 plt.figure()
152 plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
153 plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
154 plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
155 plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
156 plt.title("Training Loss and Accuracy on COVID-19 Dataset")
157 plt.xlabel("Epoch #")
158 plt.ylabel("Loss/Accuracy")
159 plt.legend(loc="lower left")
160 plt.savefig("plot.png")
```

5.10. Guardado del modelo

Por último se guarda el modelo para poder utilizarlo en un futuro sin necesidad de volver a pasar por el proceso de entrenamiento.

```
162 # serialize the model to disk
163 print("[INFO] saving COVID-19 detector model...")
164 model.save(args["model"], save_format="h5")
```

6. Resultados de la ejecución del script original

El resultado de la ejecución del script original extraído desde el propio articulo web [1], puesto que también expone la solución de la ejecución, es el siguiente:

```
1 $ python train_COVID-19.py --dataset dataset
2 [INFO] loading images...
3 [INFO] compiling model...
4 [INFO] training head...
5 Epoch 1/25
6 5/5 [==============================] - 20s 4s/step - loss: 0.7169 -
    accuracy: 0.6000 - val_loss: 0.6590 - val_accuracy: 0.5000
```

```
Epoch 2/25
5/5 [==============================] - 0s 86ms/step - loss: 0.8088 -
    accuracy: 0.4250 - val_loss: 0.6112 - val_accuracy: 0.9000
Epoch 3/25
5/5 [==============================] - 0s 99ms/step - loss: 0.6809 -
    accuracy: 0.5500 - val_loss: 0.6054 - val_accuracy: 0.5000
Epoch 4/25
5/5 [==============================] - 1s 100ms/step - loss: 0.6723 -
    accuracy: 0.6000 - val_loss: 0.5771 - val_accuracy: 0.6000
...
Epoch 22/25
5/5 [==============================] - 0s 99ms/step - loss: 0.3271 -
    accuracy: 0.9250 - val_loss: 0.2902 - val_accuracy: 0.9000
Epoch 23/25
5/5 [==============================] - 0s 99ms/step - loss: 0.3634 -
    accuracy: 0.9250 - val_loss: 0.2690 - val_accuracy: 0.9000
Epoch 24/25
5/5 [==============================] - 27s 5s/step - loss: 0.3175 -
    accuracy: 0.9250 - val_loss: 0.2395 - val_accuracy: 0.9000
Epoch 25/25
5/5 [==============================] - 1s 101ms/step - loss: 0.3655 -
    accuracy: 0.8250 - val_loss: 0.2522 - val_accuracy: 0.9000
[INFO] evaluating network...
              precision   recall f1-score  support
       covid      0.83     1.00     0.91        5
      normal      1.00     0.80     0.89        5
    accuracy                         0.90       10
   macro avg      0.92     0.90     0.90       10
weighted avg      0.92     0.90     0.90       10
[[5 0]
 [1 4]]
acc: 0.9000
sensitivity: 1.0000
specificity: 0.8000
[INFO] saving COVID-19 detector model...
```

Por lo que se puede ver en los resultados que se obtienen, el modelo predictivo construido y entrenado, es decir, la *red neuronal profunda* es bastante bueno, ya que su porcentaje de acierto sobre el conjunto de prueba es del 90 %, además si nos centramos en analizar los resultados de la matriz de confusión se puede ver:

- Los enfermos de COVID-19 los clasifica todos correctamente esto lo confirma el valor de la sensibilidad (1), lo que esto significa es que el modelo predictivo no se equivoca y si le llega una imagen de un enfermo de COVID-19 lo va a detectar.
- Las personas sanas no las detecta siempre, las detecta en un 80 % de los casos, esto lo confirma el valor de la especificidad (0.8), lo que significa que si llega

una persona sana lo detectará correctamente en un 80 % de los casos.

La gráfica conseguida en el experimento realizado en el articulo web [1] es:

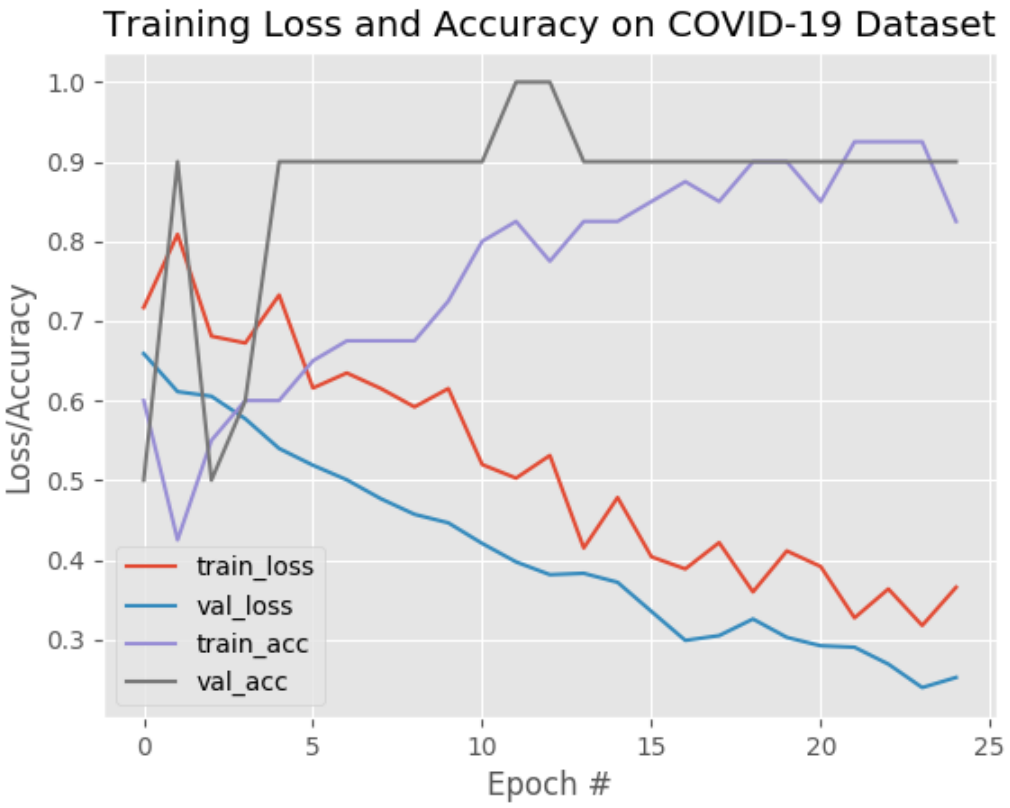

Figura 5: Gráfica de la ejecución del script original [1]

Según se puede ver en el gráfico los valores de *Accuracy* tienen a bajar mientras los valores de *loss* tienden a subir, esto como se comento anteriormente es síntoma de que el modelo predictivo está aprendiendo correctamente.

También se puede ver como el *Accuracy* de validación (en este caso es igual al *Accuracy* de test) alcanza el resultado final muy pronto y aunque tiene "un pico" con el desarrollo de las épocas, recupera el valor de 90 %.

6.1. Conclusión de resultados obtenidos

Aunque los resultados generales eran relativamente muy buenos porque llegaban al 90 %, me gustaría volver a recalcar que el modelo predictivo no acierta en el 100 % de los casos cuando se trata de predecir una persona sana, por lo que puede predecir que una persona padece COVID-19 cuando realmente esta persona está sana, y esto puede llegar a ser muy problemático puesto que a está persona se le puede poner en

cuarentena con otros pacientes con COVID-19 positivo e infectar a una persona que nunca tuvo el virus.

Aunque a priori puede parecer que sería peor que no detectara con el 100 % de precisión los enfermos de COVID-19 puesto que esto podría acabar con sus vidas, pienso que puede ser igual de grave confundirse diagnosticando que una persona tiene COVID-19 cuando realmente no es así, por lo comentado anteriormente, por lo que tendríamos que seguir mejorando el modelo para conseguir aún mejores resultados.

También se cree que con una base de datos de más imágenes, tanto de enfermos de COVID-19 como de personas sanas, se podrán conseguir un modelo predictivo bastante más sólido en la práctica. Aunque conseguir imágenes de rayos X de personas sanas es más sencillo, conseguirlas de enfermos de COVID-19 es bastante más complicado, por varios motivos, entre ellos porque estamos en plena época de COVID-19, los hospitales ya están abrumados con la cantidad de casos de COVID-19 y, dado los derechos y la confidencialidad de los pacientes, es aún más difícil reunir conjuntos de datos de imágenes médicas de calidad de manera oportuna.

Según el articulo web [1], en los próximos 12-18 meses se tendrán más conjuntos de datos de imagen COVID-19 de alta calidad.

Otra idea que se puede utilizar es que a parte de usar estás imágenes de rayos X, se utilice información adicional de la persona para entrenar este modelo como signos vitales del paciente, ubicación geográfica, la densidad de población, etc.

7. Alternativas de moficaciones

Las posibles modificaciones para intentar mejorar la *red neuronal convolucional* son infinitas, por esto después de un pequeño razonamiento previo observando como funciona el script original, las modificaciones que se piensan que pueden mejorar el rendimiento del modelo son:

- Modificar levemente la arquitectura de la red.
- Modificar el optimizador utilizado para el aprendizaje.
- Modificar la función de pérdida utilizada para el aprendizaje.
- Modificar el generador de imágenes puesto que uno de los mayores problemas como se explicaba anteriormente es la escasez de imágenes de entrenamiento, en el nuevo generador probarán diferentes combinaciones de parámetros.

8. Implementación del Script Mejorado

Después de realizar varias pruebas en la línea de lo propuesto en la sección anterior, de tal forma que se han ido probando diferentes combinaciones y parámetros,

finalmente lo que mejor ha funcionado y que por lo tanto se ha cambiado respecto al script original ha sido:

- Modificación del número de épocas utilizado.

```
# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-3
EPOCHS = 50
BS = 8
```

- Modificación leve en la arquitectura de la red de tal forma que se añade delante de la capa de salida, una nueva capa oculta será una capa densa con 32 neuronas y que utiliza la función de activación "Relu". A continuación, se puede ver la inclusión de esta capa sobre el modelo original.

```
    # load the VGG16 network, ensuring the head FC layer sets are left
    # off
    baseModel = VGG16(weights="imagenet", include_top=False,
      input_tensor=Input(shape=(224, 224, 3)))

    # construct the head of the model that will be placed on top of
       the
    # the base model
    headModel = baseModel.output
    headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
    headModel = Flatten(name="flatten")(headModel)
    headModel = Dense(64, activation="relu")(headModel)
    headModel = Dropout(0.5)(headModel)
    headModel = Dense(32, activation="relu")(headModel)
    headModel = Dense(2, activation="softmax")(headModel)

    # place the head FC model on top of the base model (this will
       become
    # the actual model we will train)
    model = Model(inputs=baseModel.input, outputs=headModel)
```

- Cambio en los parámetros del generador de imágenes, se ha añadido zoom_range = 0.1, así como se ha modificado rotation_range. A continuación, se puede ver el nuevo generador de imágenes sobre el modelo original.

```
# initialize the training data augmentation object
trainAug = ImageDataGenerator(
  rotation_range = 20,
  zoom_range = 0.1,
  fill_mode = "nearest"
   )
```

En el siguiente enlace de Github: [enlace], se puede consultar el código completo modificado por mí, que se ha ejecutado en Google Colab [2]. Los resultados obtenidos se pueden comprobar a continuación.

9. Comparaciones de resultados

Los resultado que he obtenido utilizando el script modificado por mí son:

```
[INFO] compiling model...
[INFO] training head...
Epoch 1/50
5/5 [==============================] - 1s 131ms/step - loss: 0.7188 -
    accuracy: 0.4750 - val_loss: 0.6751 - val_accuracy: 0.6000
...
Epoch 48/50
5/5 [==============================] - 1s 105ms/step - loss: 0.2408 -
    accuracy: 0.9250 - val_loss: 0.1048 - val_accuracy: 1.0000
Epoch 49/50
5/5 [==============================] - 0s 98ms/step - loss: 0.1988 -
    accuracy: 0.9500 - val_loss: 0.0901 - val_accuracy: 1.0000
Epoch 50/50
5/5 [==============================] - 1s 101ms/step - loss: 0.1012 -
    accuracy: 0.9750 - val_loss: 0.1036 - val_accuracy: 1.0000
[INFO] evaluating network...
              precision   recall f1-score  support

       covid      1.00     1.00     1.00        5
      normal      1.00     1.00     1.00        5

    accuracy                          1.00       10
   macro avg      1.00     1.00     1.00       10
weighted avg      1.00     1.00     1.00       10

[[5 0]
 [0 5]]
acc: 1.0000
sensitivity: 1.0000
specificity: 1.0000
[INFO] saving COVID-19 detector model...
```

Como se puede comprobar en los resultados se ha conseguido un modelo "perfecto", en términos de *Accuracy*, es decir, en términos de patrones bien clasificados, puesto que ha clasificado todas las imágenes del conjunto de prueba satisfactoriamente, es cierto que no había demasiadas, pero se ha conseguido el objetivo de mejorar el modelo del Script original.

A continuación en la figura 6 se muestra el resumen del *Accuracy* y del *loss* tanto del conjunto de entrenamiento como del conjunto de validación (en este caso coincide con el conjunto de prueba), a lo largo de las épocas.

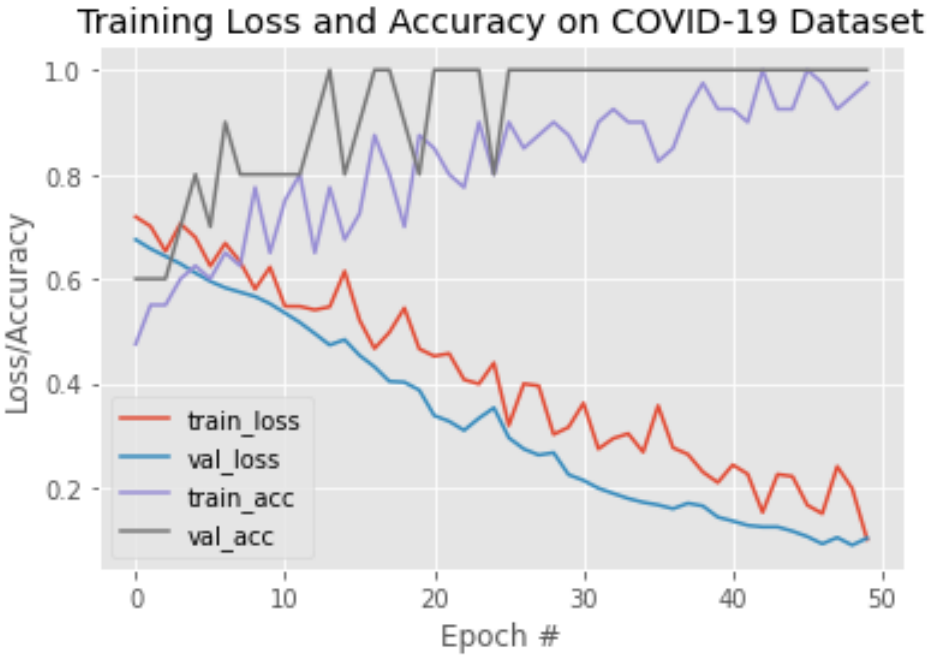

Figura 6: Gráfica de la ejecución del script modificado

En la gráfica se comprueba el satisfactorio entrenamiento del modelo. El valor máximo de porcentaje de patrones bien clasificados en el conjunto de validación se alcanza sobre la época 25 para ya no bajar en ningún momento. También se puede comprobar que aunque se aumente el número de épocas el modelo no entra en sobreentrenamiento, esto es gracias al decremento del factor de aprendizaje.

10. Más problemas médicos a resolver con *Inteligencia artificial*

En el campo de la medicina, a parte de utilizar el *Deep Learning* como predictor de enfermedades, la inteligencia artificial también se puede utilizar para otros fines como:

- **Corti:** En Copenhaguen, Dinamarca, los servicios de emergencia están utilizando este asistente de voz. Su tarea es analizar las conversaciones entre los profesionales y los pacientes y así extraer los datos. Gracias a Corti, se han salvado numerosas vidas, ya que en situaciones de emergencia, los pacientes suelen estar bajo mucha presión [7].

- **Art Medical:** Art Medical es una compañía que quiere resolver un gran problema con la medicina. En específico, el hecho de que los pacientes se enferman más en los hospitales. Art Medical desarrolló tubos de alimentación y monitores inteligentes, que actualmente están en pruebas clínicas. Estos dispositivos están hechos para evitar complicaciones fatales en pacientes que están en cuidados intensivos en los hospitales y clínicas. La empresa espera que con sus productos se puedan prevenir complicaciones que no están relacionadas con el motivo original de hospitalización. Un ejemplo de complicación es neumonía de aspiración, que es un gran riesgo que corren los pacientes intubados [8].
- **PEPPER:** es un proyecto realizado por el Centro de Investigación Biomédica en Red de Fisiopatología de la Obesidad y la Nutrición (CIBEROBN) y el Instituto de Investigación Biomédica de Girona (IDIBGI). Estos organismos, a través de la IA, desarrollaron un dispositivo portátil capaz de asesorar a los pacientes de forma personalizada, en el control de la administración de sus medicinas y el consumo de alimentos. Como resultado, el índice glucémico de las personas que implementaron el sistema mejoró considerablemente, lo que convierte a PEPPER en una herramienta de IA en medicina que resulta efectiva para el tratamiento de la diabetes tipo 1 [9].

11. Conclusiones

Las conclusiones las separaré en varias partes:

- Conclusión/Reflexión acerca del uso de estas técnicas en el campo de la medicina.
- Conclusiones acerca del trabajo.
- Conclusiones personales.

11.1. Conclusión/Reflexión acerca del uso de estas técnicas en el campo de la medicina

¿Pondrías en manos de una máquina realizar un diagnostico de alguna enfermedad significativamente grave?, esta es la pregunta que se intentará responder y justificar bajo mi punto de vista, apoyándome en opiniones de otras personas, durante esta sección del trabajo.

En primer lugar, comentar sobre las técnicas de **Deep Learning** que se ha demostrado los últimos años que funcionan muy bien en diversos problemas diferentes, no hay más que comprobar el aumento en su uso con el paso del tiempo.

En segundo lugar, cuando se entra en el campo de la medicina ya comienzan las dudas, si concretamente utilizamos el ejemplo del diagnostico de una enfermedad

como COVID-19 o Cáncer, las dudas siguen aumentando y ya cuando es un ser querido el que esta siendo analizado las dudas se trasforman generalmente en un no, puesto que debemos de pensar que los modelos predictivos se pueden equivocar, así que prefieres que ese diagnostico tan importante lo realice un médico, que tendrá los conocimientos médicos necesarios. Es importante saber que en este contexto los modelos predictivos pueden tener consecuencias muy reales, por ejemplo puede costarle la vida a nuestro ser querido, aunque son las mismas consecuencias que puede acarrear una equivocación de un médico.

Si me centro en el ejemplo de la detección del COVID-19 en imágenes de rayos X, por un lado destacar que sería posible realizarlos puesto que todos los hospitales cuentan con salas de rayos X. También sería eficiente puesto que sería un sistema de análisis automatizado para ahorrar a los profesionales médicos un tiempo valioso.

Por otro lado, como inconveniente principal podemos destacar que nada mejor como los test específicos para comprobar si padeces o no la enfermedad. Aunque la realidad no es tan sencilla como se está demostrado en muchos países del mundo durante los últimos días, no hay test suficientes para realizarle las pruebas a todo el mundo y si no muestras síntomas claros no serás una prioridad. Por lo que al utilizar estas técnicas tampoco sería necesaria la compra de los kits de prueba dedicados para detectar el COVID-19 que se hacen imposibles de costear para todas las personas en plena pandemia.

EL COVID-19 con el tiempo se normalizará y pasará a ser una enfermedad "normal", por lo que se saldrá de esta situación excepcional que se está viviendo hoy día, aún así tanto para la detección de esta enfermedad como para otra enfermedad de gravedad como el Cáncer, propongo que el *Deep Learnig* no se utilice como un predictor propiamente dicho, sino que se utilice como un sistema de apoyo a la decisión, es decir, que pueda ayudar a un médico a realizar sus diagnósticos aunque siempre prevalezca la decisión del médico, de esta manera el médico podrá contrastar o revisar su teoría, así como obtener un rápido primer opinión.

Figura 7: *Deep Learning* en la medicina

Muchos de los artículos publicados sobre este tema insisten en que no pretenden presionar para que el *Deep Learning* se utilice en el campo de la medicina sino que se realizan con fines educativos para motivar a las personas a aprender *Deep Learnign* en un área tan interesante como la medicina.

Una publicación del periódico el mundo [10], insiste en la idea que comente anteriormente, es decir, utilizar la inteligencia artificial como aliado, puesto que ha llegado para quedarse y no para sustituir a los médicos sino para ayudar y asegura que desarrollan algoritmos para analizar e interpretar imágenes, y trabajan en el aprendizaje de máquinas (machine learning) para extraer de las imágenes médicas información clínica útil. Así, la inteligencia artificial se aplica para entender mejor el desarrollo del cerebro, mejorar el diagnóstico de pacientes con demencia, que hayan sufrido un ictus o daños cerebrales, o bien realizar diagnósticos en personas con enfermedades cardiovasculares. también afirma que hay muchas cosas que se pueden hacer de forma automática para ayudar a los médicos. También añade que la inteligencia artificial es una herramienta para ofrecer diagnósticos en países en vías de desarrollo o zonas remotas donde no hay personal médico tan cualificado.

Como resumen de todo lo comentado durante esta sección, no puedo estar más de acuerdo con este artículo [10], pienso que la inteligencia artificial puede ayudar muchísimo al campo de la medicina sin la necesidad de sustituir al médico cuando no sea necesario.

11.2. Conclusiones acerca del trabajo

Las conclusiones acerca del trabajo que he obtenido son:

- El *Deep Learning* es el futuro en el campo de la medicina y ha llegado como aliado (no como sustituto) y para quedarse.
- Con técnicas de *Deep Learning* se consiguen buenos resultados con un problema tan complejo y actual como es el de la detección del COVID-19.
- Esta técnica se podría utilizar realmente, tan solo bastaría con realizar una imagen del pulmón de rayos X por sospechoso de contagio.
- Conforme aumente el número de imágenes de rayos X con el que se puede trabajar, se irán obteniendo modelos más robustos.

11.3. Conclusiones personales

Las conclusiones personales que he obtenido son:

- El *Deep Learning* me parece cada vez más interesante y pienso que será el futuro en casi todas las áreas, ya sea de una manera u de otra ha venido para quedarse en nuestras vidas.
- Gracias a este trabajo he tenido tiempo para dedicarle a un tema que tenía ganas de indagar hace tiempo, es decir, los problemas de medicina que se pueden resolver utilizando este tipo de técnicas, así como otras técnicas de inteligencia artificial.

Referencias

[1] *Pyimagesearch - https://www.pyimagesearch.com/2020/03/16/detecting-covid-19-in-x-ray-images-with-keras-tensorflow-and-deep-learning/*. URL: https://www.pyimagesearch.com/2020/03/16/detecting-covid-19-in-x-ray-images-with-keras-tensorflow-and-deep-learning/. [En Línea. Última consulta: 06-05-2020].

[2] *Google Colaboratory*. URL: https://colab.research.google.com/notebooks/intro.ipynb#recent=true. [En Línea. Última consulta: 06-05-2020].

[3] *Google Drive*. URL: https://drive.google.com/. [En Línea. Última consulta: 06-05-2020].

[4] *OpenCV*. URL: https://opencv.org/. [En Línea. Última consulta: 06-05-2020].

[5] *Github - covid-chestxray-dataset*. URL: https://github.com/ieee8023/covid-chestxray-dataset. [En Línea. Última consulta: 06-05-2020].

[6] *Kaggle - Chest X-Ray Images (Pneumonia)*. URL: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia. [En Línea. Última consulta: 06-05-2020].

[7] *Master Deep Learning*. URL: https://master-deeplearning.com/uso-deep-learning-medicina/. [En Línea. Última consulta: 06-05-2020].

[8] *Enter - Ejemplos de inteligencia artificial*. URL: https://www.enter.co/chips-bits/apps-software/ejemplos-inteligencia-artificial-salud/. [En Línea. Última consulta: 06-05-2020].

[9] *Enzyme - Inteligencia artificial en medicina: casos reales de éxito*. URL: https://blog.enzymeadvisinggroup.com/inteligencia-artificial-en-la-medicina. [En Línea. Última consulta: 06-05-2020].

[10] *El mundo - Un nuevo aliado de los médicos*. URL: https://lab.elmundo.es/inteligencia-artificial/salud.html. [En Línea. Última consulta: 06-05-2020].