Practica 2: Predicción de Neumonía

10 de julio de 2023

Dataset: Rayos X de Pulmones. Disponible en:
https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

1. Objetivo de la práctica.

¿Podemos clasificar correctamente radiografías de pulmones de pacientes que tienen neumonía y de pacientes que tienen pulmones sanos?

2. Conceptos.

Ya se habló de las librerías en la Practica 1, usaremos las mismas para poder reforzar el análisis de problemas de clasificación binaria

El dataset consiste en 2 carpetasb train y test que a su vez cuentan con 2 subcarpetas: NORMAL y PNEUMONIA, que contienen en total 186 elementos.

3. Herramientas a usar.

- Computadora con acceso a internet.
- Los siguientes programas y librerías:
  - Python versión 3.8.8
  - Anaconda versión 4.10.1
  - Jupyter-lab 3.014
  - NumPy versión 1.21.4
  - Pandas versión 1.2.4
  - Matplotlib versión 3.3.4
  - Zipfile versión 3.2.0
  - Split-folders 0.5.1

4. Desarrollo.

4.1. Entender el problema.

Se tiene un dataset donde se encuentran imágenes rayos X con pulmones que están sanos y pulmones con síntomas de neumonía. Ambos a simple vista para una persona normal parecen ser iguales pero una de las características de los pulmones que tiene neumonía es que presentan manchas blancas. Unas son muy claras de ver en las radiografías y otras no tanto. El objetivo de esta práctica es aplicar los conocimientos de la Práctica 1 y practicar con este nuevo dataset.

4.2. Criterio de evaluación.

Usaremos la exactitud (accuracy)

4.3. Preparar los datos.

Debemos importar todas las librerías con las que estaremos trabajando durante la práctica.

```
import os
import random
import zipfile
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Estaremos trabajando con Tensorflow, por lo que es necesario comprobar que se esté usando nuestra GPU.

```
tf.config.list_physical_devices('GPU')


[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Aquí definimos la ruta base de nuestros datos. Además, extraeremos las carpetas de los archivos y comprobaremos que se encuentren ahí.

```
Path = "../Practica 2/" #Depende de la ruta en tu m quina

with zipfile.ZipFile('../Practica 2/archive.zip', 'r') as archive:
    # Extrae todo el contenido del achivo ZIP en el directorio
     actual
    archive.extractall("../Practica 2/")

os.listdir("../Practica 2/xray_dataset_covid19/")

['test', 'train']
```

A diferencia de la Practica 1, los archivos ya están divididos en conjuntos de test y train, por lo que podemos avanzar hasta el análisis de las imágenes haciendo uso de Matplotlib y las gráfica.

```
covid_path = "../Practica 2/xray_dataset_covid19/"

train_normal = covid_path+"train/NORMAL/"
train_pneumonia = covid_path+"train/PNEUMONIA/"

train_dir_normal = os.listdir(train_normal)
train_dir_pneumonia = os.listdir(train_pneumonia)
```

```
plt.figure(figsize=(15,10))
for i in range(12):
    ran = random.choice(range(1,70))
    rand_normal = [os.path.join(train_normal, f) for f in
    train_dir_normal[ran:ran+1]]
    rand_pneumonia = [os.path.join(train_pneumonia, f) for f in
    train_dir_pneumonia[ran:ran+1]]
    rand = random.choice(rand_pneumonia+rand_normal)
    name = rand.split('/')
    name = name[-2] + ' Lungs'
    plt.subplot(3, 4, i+1)
    img = plt.imread(rand)
    plt.imshow(img,cmap = 'gray')
    plt.axis(False)
    plt.title(name)
plt.show()
```

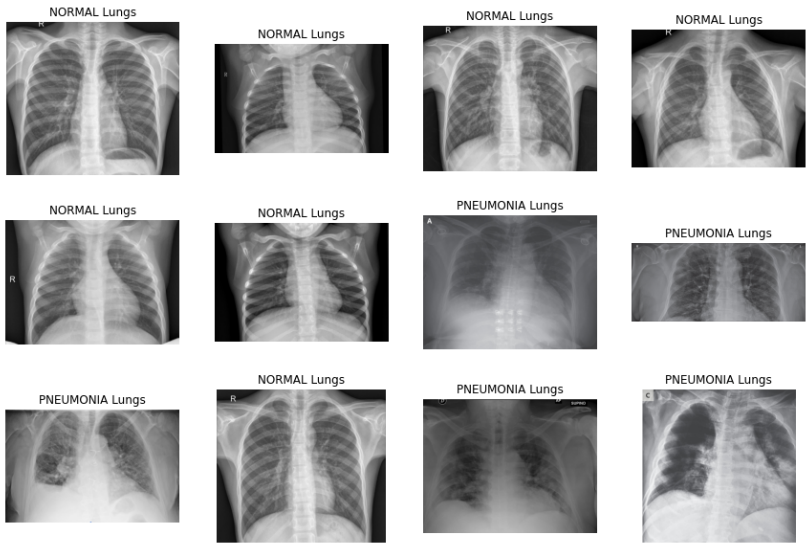

Figura 1: Gráfica con subgráficas presentando una muestra de las imágenes de las carpetas.

Las imágenes son variadas, lo cuál es lo que buscábamos. En unas radiografías es obvio la presciencia de neumonía y otras ameritan la creación de una red para poder ayudar a la detección de esta enfermedad. A continuación repetiremos el proceso de sacar las dimensiones de una imagen para poder ayudar al modelo.

```
sample = random.choice(train_dir_normal)
sample = plt.imread(train_normal + sample)
samp = sample.shape
plt.imshow(sample)
print('Sample_Image_size(H*W):', samp)
```

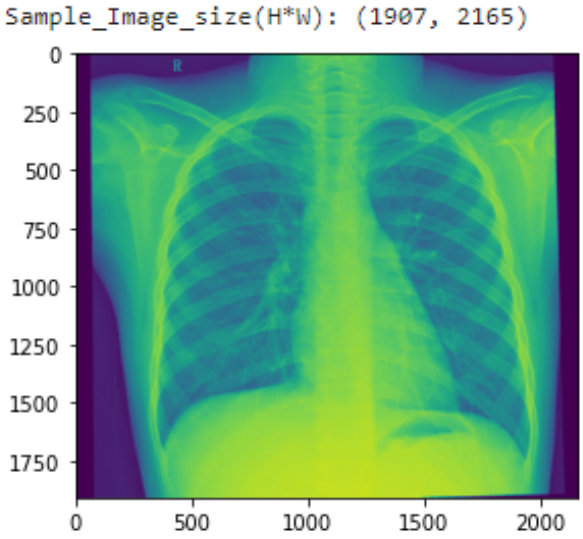

Figura 2: Gráfica con una radiografía sin escalado a grises.

Crearemos el DataFrame para extraer las características promedio de las imágenes.

```
x, y = [], []
for i in train_dir_normal:
    img_array = plt.imread(train_normal + i)
    shape = img_array.shape
    x.append(shape[0])
    y.append(shape[1])
df_shape = pd.DataFrame({'height': x, 'width': y, 'index': range(0,
     len(train_dir_normal))})
```

```
df_shape.head()

    height  width index
0   1317    1857     0
1   1509    2111     1
```

```
2    1837    2031      2
3    1326    1663      3
4    1818    2053      4
```

```
df_shape.describe()

              height          width        index
count      74.000000      74.000000    74.000000
mean     1539.689189    1968.094595    36.500000
std       448.372030     340.261590    21.505813
min       617.000000    1240.000000     0.000000
25%      1255.000000    1765.000000    18.250000
50%      1469.500000    1959.500000    36.500000
75%      1870.750000    2150.750000    54.750000
max      2713.000000    2752.000000    73.000000
```

La imágenes de este DataFrame, aunque son menos que las de la práctica anterior, cuentan con dimensiones más grandes, por lo que eso podría ayudar a extraer características más fácil en el entrenamiento de nuestra red.

4.4. Construir el Modelo.

Comenzaremos normalizando los valores de las imágenes para que sean binarios; solo 0 y 1. Esto porque es lo que queremos en realidad; un modelo de clasificación binaria.

```
#Normalizan los valores entre 0 y 1
train = ImageDataGenerator(rescale=1./255)
test = ImageDataGenerator(rescale=1./255)
```

Ahora vamos a generar los dataset de entrenamiento y prueba. Estas vamos a cargarlas haciendo las imágenes de un tamaño estándar cuadrado de 400 por 400 píxeles. Esto porque, aunque vimos que hay un tamaño estándar más grande, las imágenes más grandes causan que el entrenamiento tarde más si es que nuestro equipo de computo no es muy potente. Estableceremos que las imágenes de pasen con un filtro de escala a grises para poder resaltar mejor los puntos que señalan la neumonía y ayudar a la red a no tener información que pueda "meter ruido.[a]l entrenamiento. El tamaño de batch es la cantidad de imágenes que pasará por paso de época durante el entrenamiento y el modo de clasificación es puesto en binario.

```
tr_dir = covid_path+"train/"
te_dir = covid_path+"test/"

train = train.flow_from_directory(covid_path+"train/",
                                  target_size = (400,400),
                                  color_mode = 'grayscale',
                                  batch_size = 10,
                                  class_mode = 'binary')

test = test.flow_from_directory(covid_path+"test/",
                                target_size = (400,400),
                                color_mode = 'grayscale',
                                batch_size = 10,
```

```
                                    class_mode= 'binary')

Found 148 images belonging to 2 classes.
Found 40 images belonging to 2 classes.
```

```
test.class_indices
{'NORMAL': 0, 'PNEUMONIA': 1}
```

Usaremos una estructura similar a la de la Práctica 1 usando 4 capas de Conv2D con Maxpool2D para ver si, al ser imágenes más grandes, podemos evitar el sobre-ajuste que sufrimos en con las imágenes de mosquitos.

```
model = keras.models.Sequential([
    keras.layers.Conv2D(32, (5,5), activation = 'relu', input_shape
     = (400,400, 1)),
    keras.layers.MaxPool2D((2,2)),

    keras.layers.Conv2D(64, (3,3), activation = 'relu'),
    keras.layers.MaxPool2D(2,2),
    keras.layers.Conv2D(128, (3,3), activation = 'relu'),
    keras.layers.MaxPool2D((2,2)),

    keras.layers.Conv2D(256, (3,3), activation = 'relu'),
    keras.layers.MaxPool2D((2,2)),

    keras.layers.Flatten(),

    keras.layers.Dense(512, activation = 'relu'),

    keras.layers.Dense(1, activation = 'sigmoid')
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 396, 396, 32)      832

 max_pooling2d (MaxPooling2D  (None, 198, 198, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 196, 196, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 98, 98, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 96, 96, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 48, 48, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 46, 46, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 23, 23, 256)      0
 2D)

 flatten (Flatten)           (None, 135424)            0

 dense (Dense)               (None, 512)               69337600

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 69,726,465
Trainable params: 69,726,465
Non-trainable params: 0
_________________________________________________________________
```

Figura 3: Esta es la estructura de la red neuronal con la que se van a entrenar con las imágenes.

Ya con la estructura de la red creada solo nos queda mandar a llamar el entrenamiento. Este lo realizaremos haciendo uso de la función de model.fit donde se le dirá a la red con qué imágenes se le alimentará, cuántas veces tendrá que hacer el entrenamiento y qué imágenes usará para corroborar si su clasificación es correcta.

```
model.compile(optimizer="adam",
              loss='binary_crossentropy',
              metrics = ['accuracy'])

history = model.fit(
            train, # pass in the training generator
            steps_per_epoch=7,
            epochs=30,
```

```
            validation_data=test, # pass in the validation
    generator
            validation_steps=2,
            verbose=2
            )
```

```
Epoch 15/30
7/7 - 2s - loss: 0.1123 - accuracy: 0.9429 - val_loss: 0.0061 - val_accuracy: 1.0000 - 2s/epoch - 290ms/step
Epoch 16/30
7/7 - 2s - loss: 0.0804 - accuracy: 0.9714 - val_loss: 0.2341 - val_accuracy: 0.9000 - 2s/epoch - 282ms/step
Epoch 17/30
7/7 - 2s - loss: 0.0887 - accuracy: 0.9853 - val_loss: 0.3259 - val_accuracy: 0.9500 - 2s/epoch - 283ms/step
Epoch 18/30
7/7 - 2s - loss: 0.0472 - accuracy: 1.0000 - val_loss: 0.0086 - val_accuracy: 1.0000 - 2s/epoch - 334ms/step
Epoch 19/30
7/7 - 2s - loss: 0.0676 - accuracy: 0.9857 - val_loss: 0.0021 - val_accuracy: 1.0000 - 2s/epoch - 290ms/step
Epoch 20/30
7/7 - 2s - loss: 0.1089 - accuracy: 0.9559 - val_loss: 0.1582 - val_accuracy: 0.9500 - 2s/epoch - 308ms/step
Epoch 21/30
7/7 - 2s - loss: 0.0383 - accuracy: 0.9853 - val_loss: 0.1650 - val_accuracy: 0.9500 - 2s/epoch - 310ms/step
Epoch 22/30
7/7 - 2s - loss: 0.0191 - accuracy: 1.0000 - val_loss: 0.4464 - val_accuracy: 0.9000 - 2s/epoch - 282ms/step
Epoch 23/30
7/7 - 2s - loss: 0.0442 - accuracy: 0.9714 - val_loss: 0.0967 - val_accuracy: 0.9500 - 2s/epoch - 235ms/step
Epoch 24/30
7/7 - 2s - loss: 0.0060 - accuracy: 1.0000 - val_loss: 0.0048 - val_accuracy: 1.0000 - 2s/epoch - 279ms/step
Epoch 25/30
7/7 - 2s - loss: 0.2338 - accuracy: 0.9559 - val_loss: 1.3648 - val_accuracy: 0.8500 - 2s/epoch - 252ms/step
Epoch 26/30
7/7 - 2s - loss: 0.3981 - accuracy: 0.9286 - val_loss: 0.0168 - val_accuracy: 1.0000 - 2s/epoch - 279ms/step
Epoch 27/30
7/7 - 2s - loss: 0.3849 - accuracy: 0.8571 - val_loss: 0.0092 - val_accuracy: 1.0000 - 2s/epoch - 267ms/step
Epoch 28/30
7/7 - 2s - loss: 3.5054 - accuracy: 0.7941 - val_loss: 0.0031 - val_accuracy: 1.0000 - 2s/epoch - 266ms/step
Epoch 29/30
7/7 - 2s - loss: 0.1829 - accuracy: 0.9118 - val_loss: 0.0915 - val_accuracy: 0.9500 - 2s/epoch - 322ms/step
Epoch 30/30
7/7 - 2s - loss: 0.2569 - accuracy: 0.9265 - val_loss: 0.1272 - val_accuracy: 0.9500 - 2s/epoch - 280ms/step
```

Figura 4: Aquí se ve el progreso del entrenamiento durante las últimas 15 épocas. Podemos ver cómo en la parte de validación la exactitud comienza a sobre-ajustarse

Simplemente con el resultado del entrenamiento podemos ver que tenemos problemas de sobre-ajuste, ya que los valores de perdida y exactitud en las últimas épocas en el conjunto de validación la diferencia y variación de estos es poca o nula.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

# plot accuracy with matplotlib
plt.plot(epochs, acc)
plt.plot(epochs, val_acc)
plt.title('Accuracy in training and validation')
plt.figure()

# plot loss with matplotlib
```

```
15 plt.plot(epochs, loss)
16 plt.plot(epochs, val_loss)
17 plt.title('Loss in training and validation')
```

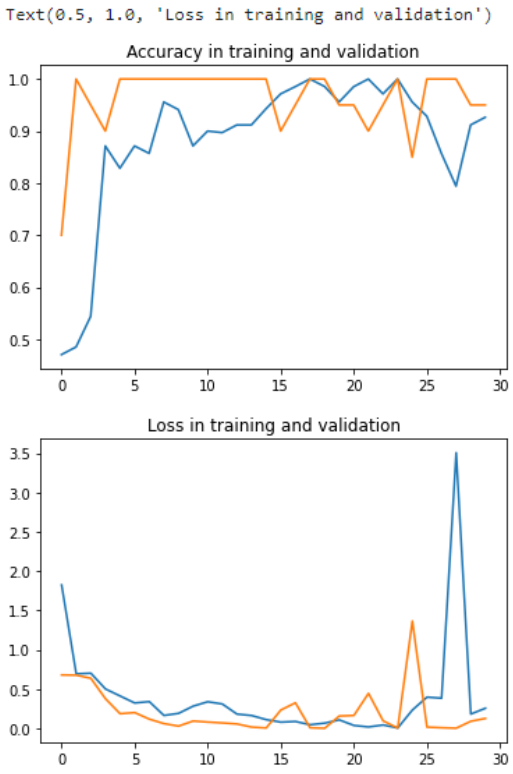

Figura 5: Las gráficas que muestran la exactitud y pérdida del entrenamiento del modelo en el conjunto de entrenamiento y validación.

4.5. Análisis de errores.

Como se puede ver, la clasificación del modelo es muy buena hasta cierto momento y luego se dispara cuando comienza a dar clasificaciones no correspondientes. Al igual que el modelo anterior, podemos tratar de ver si la predicción de alguna imagen puede ser correcta.

```
def predictImage(filename):
    img1 = image.load_img(filename,target_size=(400,400),
    color_mode="grayscale")

    plt.imshow(img1)

    Y = image.img_to_array(img1)

    X = np.expand_dims(Y,axis=0)
    val = model.predict(X)
    print(val)
    if val == 1:

        plt.xlabel("PNEUMONIA",fontsize=30)


    elif val == 0:

        plt.xlabel("NORMAL",fontsize=30)

predictImage(Path + "pneumonia-test.jpg")
```

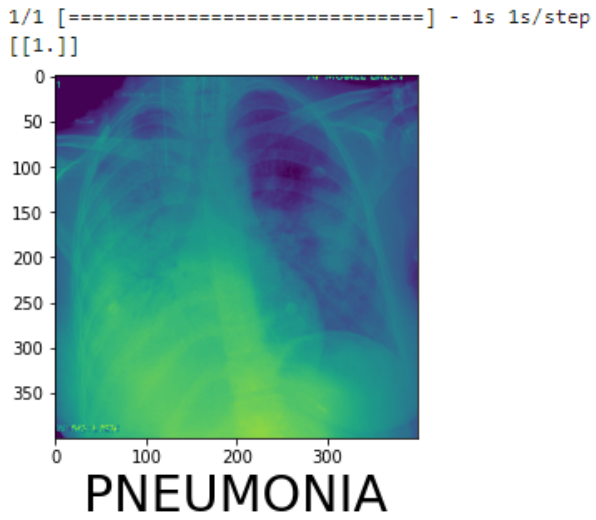

Figura 6: La imagen fue cargada al modelo y la predicción resultante logró ser acertada.

Aunque los resultados de la predicción y el entrenamiento del modelo resultaron ser satisfactorios, es puntual señalar que la cantidad de imágenes con las que se trabajo es reducida, por lo que nos estamos frente a un posible caso de sobre-ajuste. Esto se da por sobre-entrenar la red y es causado por la cantidad limitada de información para trabajar.

Es necesario buscar una fuente de imágenes más grande o encontrar un método para aumentar la cantidad de datos para presentarlos a la red.

4.6. Implementación.

Consulta el Notebook de Jupyter en la carpeta de **Practica 2** llamado ***TF-Pulmones.ipynb***