Actividad

Descripción ¶ (file:///C:/Users/crist/Downloads/Maria_de_Madaria_Lopez.html#Descripci%C3%B3n)

El uso de radiografías en el ámbito medico esta ampliamente extendido. Los radiologos son los médicos que analizan las imágenes y redactan el informe sobre lo que ven. Desde hace algunos años, los algoritmos de machine learning, en especial la redes neuronales artificiales, están analizando imágenes con un gran éxito. En esta PEC se plantea la implementación de una red neuronal convolucional para clasificar radiografías en normales o con derrame. El conjunto de datos (ver zip) que vamos a utilizar consta de 350 radiografías de tórax normales (carpeta con el nombre normal) y 350 radiografías con derrame (carpeta con el nombre effusion), tomadas y seleccionadas del conjunto de datos público NIH ChestXray14: https://www.nih.gov/news-events/news-releases/nih-clinical-center-provides-one-largest-publicly_available-chest-x-ray-datasets-scientific-community (https://www.nih.gov/news-events/news-releases/nih-clinical-center-provides-one-largest-publicly%02available-chest-x-ray-datasets-scientific-community). Como ejemplo, se presenta la imagen del fichero n99.png del grupo normal.

Empezaremso pudiendo converir el ipynb a pdf con una librerias de nbestencion

In [54]:
```
!jupyter nbconvert --to html
```

```
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include the error message in the
cell output (the default behaviour is to abort conversion). This flag is only relevant if '--execute' was spe
cified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
```

```
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWrite
r.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWrite
r.build_directory= --ClearOutputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.exclude_output_prompt=Tru
e]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporter.exclude_input=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python', 'rst', 'script',
'slides']
            or a dotted object name that represents the import path for an
            `Exporter` class
    Default: 'html'
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template file to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_file]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
```

```
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                    to output to the directory of each notebook. To recover
                                    previous default behaviour (outputting to the current
                                    working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

                > jupyter nbconvert mynotebook.ipynb
```

```
            which will convert mynotebook.ipynb to the default format (probably HTML).

            You can specify the export format with `--to`.
            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python',
'rst', 'script', 'slides'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX includes
            'base', 'article' and 'report'.  HTML includes 'basic' and 'full'. You
            can specify the flavor of the format used.

            > jupyter nbconvert --to html --template basic mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```

Enunciado

1. Desempaquetar y leer el fichero de datos

Empezaremos uniendo nuestro archivo ipynb a nuestro google drive, donde tenemos nuestras imagenes de derrame pulmonar

In [3]:
```
# Necesitaremos montar su disco usando los siguientes comandos:
# Para obtener más información sobre el montaje, puedes consultar: https://stackoverflow.com/questions/46986398/import-data-into-google-colaboratory

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Cargamos las librerias que vamos a usar

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import os
import random
import tensorflow as tf
from tensorflow import keras
import keras.backend as K
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers import SeparableConv2D, ZeroPadding2D, Conv2D, MaxPool2D, BatchNormalization
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras.optimizers import Adam
import matplotlib.image as mpimg
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, confusion_matrix
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

Cargamos nuestra carpeta con las imagenes de effusion y de pacientes normales

In [5]:
```
inputdir = "/content/drive/MyDrive/Pec_CNN/"
path_imagen = inputdir + "imagenes"
imagen_normal = os.path.join(path_imagen + '/normal')
imagen_effusion = os.path.join(path_imagen + '/effusion')
```

Imprimimos, cuantas imagenes tenemos de cada carpeta tanto las imagenes de pacientes normales, como pacientes patologicos

In [6]:
```
print("Total number of normal images in training set: ",len(os.listdir(imagen_normal)))
print("Total number of effusion images in training set: ",len(os.listdir(imagen_effusion)))
```

```
Total number of normal images in training set:  350
Total number of effusion images in training set:  350
```

Como vemos tenemos 350 imagenes en la carpeta de pacientes sanos y 350 imagenes en la carpeta de pacientes effusion

2. Presentar algunas imágenes de radiografías normales y con derram

Vamos a imprimir unas imagenes tanto de pacientes sanos, como pacientes effusion(con derrame)

In [7]:
```
# Parámetros para nuestro gráfico; generaremos imágenes en una configuración 4x4
nrows = 4
ncols = 4
# Set up matplotlib fig, and size it to fit 4x4 pics
fig = plt.gcf()
fig.set_size_inches(ncols * 4, nrows * 4)
next_normal_pix = [os.path.join(imagen_normal, fname) for fname in os.listdir(imagen_normal)[0:8]]
next_effusion_pix = [os.path.join(imagen_effusion, fname) for fname in os.listdir(imagen_effusion)[0:8]]
for i, img_path in enumerate(next_normal_pix+next_effusion_pix):
    # Set up subplot; subplot indices start at 1
    sp = plt.subplot(nrows, ncols, i + 1)
    sp.axis('Off') # No mostrar ejes (o líneas de cuadrícula)
    img = mpimg.imread(img_path)
    if i<=7 :
        sp.title.set_text('NORMAL')
    else :
        sp.title.set_text('EFFUSION')
    plt.imshow(img)
plt.show()
```

NORMAL

NORMAL

NORMAL

NORMAL

NORMAL

NORMAL

NORMAL

NORMAL

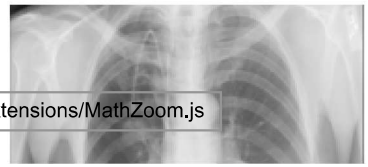
EFFUSION

EFFUSION

EFFUSION

EFFUSION

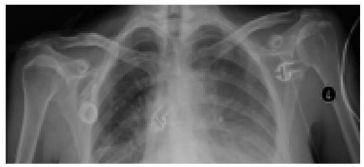
EFFUSION

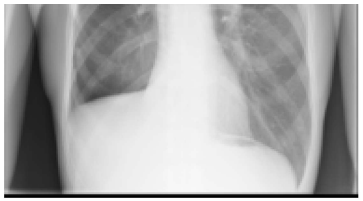
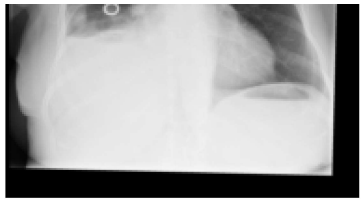
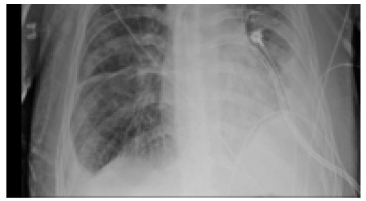


EFFUSION

EFFUSION

EFFUSION




Como podemos ver en las imagenes, son bastante llamativas las diferencias entre pacientes normales y con derramen(effusion)

In [8]:
```
next_normal_pix,next_effusion_pix## Veo las imagenes que he impreso
```

Out[8]:
```
(['/content/drive/MyDrive/Pec_CNN/imagenes/normal/n10.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/normal/n114.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/normal/n119.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/normal/n112.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/normal/n123.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/normal/n104.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/normal/n107.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/normal/n11.png'],
 ['/content/drive/MyDrive/Pec_CNN/imagenes/effusion/e108.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/effusion/e109.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/effusion/e1.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/effusion/e105.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/effusion/e100.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/effusion/e104.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/effusion/e103.png',
  '/content/drive/MyDrive/Pec_CNN/imagenes/effusion/e107.png'])
```

Miro que contenga 512 pixeles, tanto de largo como de ancho y 3 canales, que reducire en el siguiente aspartado a 1, ya que es una imagen en blanco y negro

In [9]:
```
imagen_normal_10 = cv2.imread("/content/drive/MyDrive/Pec_CNN/imagenes/normal/n10.png")
imagen_effusion_108= cv2.imread("/content/drive/MyDrive/Pec_CNN/imagenes/effusion/e108.png")
```

Miro que contenga 512 pixeles, tanto de largo como de ancho y 3 canales, que reducire en el siguiente aspartado a 1, ya que es una imagen en blanco y negro

In [10]:
```
imagen_normal_10.shape, imagen_effusion_108.shape
```

Out[10]:
```
((512, 512, 3), (512, 512, 3))
```

Como vemos todo esta correcto

3. Preprocesar las imágenes originales de 512x512x3 para convertirlas 64x64x1.

Solo se coge un canal de los 3 canales originales porque las imágenes son en blanco y negro. Todos los canales tienen la misma información

Crearemos una funcion que nos permita etiquetar las imagenes, y cambiar el tamaño de estas de 512 a 64, y tambien, tendremos un solo canal.

In [11]:
```
def loadImages(path, urls, target):  # Colocamos las etiquetas
  images = []       # Tuna lista vacia que contendra las imagenes
  labels = []       # Otra lista vacia que contendra las etiquetas

  for i in range(len(urls)):
    img_path = path + "/" + urls[i] # Esto dara lugar a leer las imagenes una a una
    img = cv2.imread(img_path, 0)   # Al poner 0 cambiamos el canal
    img = cv2.resize(img, (64, 64)) # Cambiamos las dimensiones de las imagenes
    images.append(img)              # Iremos añadiendo las imagenes una a una
    labels.append(target)           # Ponem os la ertiqueta de la imagen
  images = np.asarray(images)
  return images, labels
```

In [12]:
```
imagen_normal = os.path.join(path_imagen + '/normal')
imagen_effusion = os.path.join(path_imagen + '/effusion')
```

In [13]:
```
normal_directory= os.listdir(imagen_normal)
effusion_directory= os.listdir(imagen_effusion)
```

In [14]:
```
print("Total number of normal images in training set: ",len(os.listdir(imagen_normal)))
print("Total number of effusion images in training set: ",len(os.listdir(imagen_effusion)))
```

```
Total number of normal images in training set:  350
Total number of effusion images in training set:  350
```

Vale ahora voy a introducir la funcion loadImagem, dentro de imge_normal junto con normal_directory y señalare la etiqueta que corresponde con paciente sano

In [15]:
```
Imagen_Normal, TargetNormal = loadImages(imagen_normal,normal_directory, 0)
```

Hacemos lo mismo con las imagenes de pacientes de hedema, pero cambiasmos la etiqueta a 1

In [16]:
```
Imagen_Effusion, TargetEffusion = loadImages(imagen_effusion,effusion_directory, 1)
```

In [17]:
```
len(normal_directory)
```

Out[17]: 350

In [18]:
```
len(effusion_directory)
```

Out[18]: 350

In [19]:
```
len(Imagen_Normal)
```

Out[19]: 350

In [20]:
```
len(Imagen_Effusion)
```

Out[20]: 350

Comprobamos con la Imagen_Normal, que el tamaño de nuestra foto ha cambiado, ya no tenemos fotos de 512, sino tenemos fotos de 64 pixeles, , tambien podemos verlo en las imagenes de Effusion o Derrame

```
In [21]: Imagen_Normal.shape,Imagen_Effusion.shape
```

```
Out[21]: ((350, 64, 64), (350, 64, 64))
```

4. Normalizar mediante la transformación min-max las imágenes

Podemos normalizar los datos de dos formas, o con el min y maxino que tenemos un ejemplo https://gitlab.uoclabs.uoc.es/machinelearningbioinformatics/machine-learning-bioinformatics/-/blob/master/5_NeuralNetworks/CNN_MNIST.ipynb (https://gitlab.uoclabs.uoc.es/machinelearningbioinformatics/machine-learning-bioinformatics/-/blob/master/5_NeuralNetworks/CNN_MNIST.ipynb), o dividiendo entre 255 que es el numero maximo de pixeles. Empezaremos con la funcion np.r que es una permutación que se refiere al proceso de organizar todos los individuos de un conjunto dado para formar una secuencia.

```
In [22]: datos= np.r_[Imagen_Normal,Imagen_Effusion]
```

Estoy organizando en un mismo conjunto de datos que llamo datos tanto las imagenes de Imaggen_Normal con Imagen_effusion, y como puedo comprobar es la suma de ambas caropetas con un numero de 700 imagenes con un tamaño de 64pixels cada una

```
In [23]: datos.shape
```

```
Out[23]: (700, 64, 64)
```

Hacemos lo mismo con las etiquetas, que previamente hemos concatenado, y obtenemos 700 etiquetas

In [24]:
```
targets = np.r_[TargetNormal,TargetEffusion]
```

In [25]:
```
targets.shape
```

Out[25]:
```
(700,)
```

Aqui lo que vamos a hacer es sacar el maximo y el minimo, en la pregunta nos indica, lo saquemos con el maxim y el minimo, pongo un link de un repositorio de uno de los docentes, donde nos explica como se realiza https://gitlab.uoclabs.uoc.es/machinelearningbioinformatics/machine-learning-bioinformatics/-/blob/master/5_NeuralNetworks/CNN_MNIST.ipynb (https://gitlab.uoclabs.uoc.es/machinelearningbioinformatics/machine-learning-bioinformatics/-/blob/master/5_NeuralNetworks/CNN_MNIST.ipynb)

In [26]:
```
# Encontramos los valores minimos y maximos para el set del training
v_min = np.min(datos)
v_max = np.max(datos)
print("Min. and max. values before normalization are {} and {}.".format(v_min, v_max))

# Normalizamos
datos_Norm = (datos - v_min) / (v_max - v_min)

print("Min. and max. values after normalization are {} and {}.".format(np.min(datos_Norm), np.max(datos_Norm
)))
```

```
Min. and max. values before normalization are 0 and 255.
Min. and max. values after normalization are 0.0 and 1.0.
```

Vamos a imprimir la base normalizada y concatenada, para comprobar que los datos estan entre 1 y 0

In [27]:
```
print(datos_Norm)
```

```
[[[0.02745098 0.02745098 0.05882353 ... 0.03529412 0.03921569 0.0745098 ]
  [0.02352941 0.1372549  0.19215686 ... 0.03137255 0.03529412 0.06666667]
  [0.15294118 0.24313725 0.3254902  ... 0.11372549 0.03137255 0.05882353]
  ...
  [0.82745098 0.88235294 0.89411765 ... 0.61960784 0.67058824 0.85098039]
  [0.82352941 0.8745098  0.89803922 ... 0.43529412 0.57647059 0.77254902]
  [0.84313725 0.88627451 0.91372549 ... 0.42745098 0.56470588 0.68627451]]

 [[0.01568627 0.01568627 0.01568627 ... 0.01176471 0.01176471 0.01176471]
  [0.01568627 0.01568627 0.01176471 ... 0.01176471 0.01176471 0.01176471]
  [0.01176471 0.01176471 0.01176471 ... 0.01176471 0.01176471 0.01176471]
  ...
  [0.2745098  0.07843137 0.00784314 ... 0.01176471 0.01568627 0.01176471]
  [0.21176471 0.05882353 0.00784314 ... 0.01176471 0.01176471 0.01176471]
  [0.17254902 0.03529412 0.00784314 ... 0.01176471 0.01176471 0.01176471]]

 [[0.0745098  0.07058824 0.06666667 ... 0.16862745 0.03529412 0.03921569]
  [0.07058824 0.07058824 0.0627451  ... 0.2745098  0.20392157 0.11764706]
  [0.0627451  0.0627451  0.04705882 ... 0.51764706 0.2745098  0.21176471]
  ...
  [0.16078431 0.30588235 0.41568627 ... 0.54509804 0.41568627 0.25882353]
  [0.15686275 0.29411765 0.40392157 ... 0.5372549  0.39607843 0.23137255]
  [0.15294118 0.2627451  0.38431373 ... 0.52156863 0.38039216 0.21960784]]

 ...

 [[0.         0.         0.         ... 0.         0.         0.        ]
  [0.         0.         0.         ... 0.         0.         0.        ]
  [0.         0.00392157 0.05098039 ... 0.         0.         0.        ]
  ...
  [0.27058824 0.35294118 0.40784314 ... 0.18039216 0.09803922 0.05882353]
  [0.27843137 0.36078431 0.41960784 ... 0.2        0.14901961 0.05490196]
  [0.34901961 0.4        0.43921569 ... 0.21960784 0.16862745 0.05490196]]

 [[0.02745098 0.03137255 0.03137255 ... 0.03137255 0.03137255 0.03137255]
  [0.30980392 0.03137255 0.03137255 ... 0.03137255 0.03137255 0.03137255]
  [0.05882353 0.03137255 0.03137255 ... 0.03137255 0.03137255 0.03137255]
  ...
  [0.40784314 0.38431373 0.35294118 ... 0.05882353 0.06666667 0.06666667]
  [0.38823529 0.38431373 0.35294118 ... 0.0627451  0.0627451  0.06666667]
  [0.36078431 0.43137255 0.37254902 ... 0.0627451  0.07843137 0.06666667]]
```



```
 [[0.00784314 0.00784314 0.00784314 ... 0.01176471 0.01176471 0.01176471]
```

```
[0.00784314 0.00784314 0.00784314 ... 0.01176471 0.01176471 0.01176471]
[0.00784314 0.00784314 0.00784314 ... 0.01176471 0.01176471 0.01176471]
...
[0.08235294 0.01176471 0.21568627 ... 0.87058824 0.87058824 0.84313725]
[0.12156863 0.01960784 0.39215686 ... 0.87843137 0.8745098  0.85882353]
[0.02745098 0.49019608 0.43529412 ... 0.89803922 0.89411765 0.87058824]]]
```

Como comprobamos los vvalores que tenemos estan normalizados entre 1 y 0

5. Dividir el conjunto de datos en 600 train y 100 de test. Tratar de equilibrar las dos clases.

Dividiremos el dataset como nos indica la pregunta 5, para eso usaaremos la libreria se sckit_learn https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html), esta libreria nos permite, dividir en entrenamnientos y test los datos que hemoc ocncatenado y normalizxado en el punto anterior

In [28]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(datos_Norm, targets, test_size=0.1428)
```

Miraremos con la funcion shape, como se ha dividido nuestras base normalizada tanto en trenamiento como en test, y dividiremos x_train como x_test para entrenar y tendra toda la informacion de todas las imagenes tanto de pacientes normales, como de pacientes co hemorragia o (effusion)

In [29]:
```
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

Out[29]:
```
((600, 64, 64), (600,), (100, 64, 64), (100,))
```

Como vemos coincide a la perfeción con lo que se nos pide en el enunciado, las partes de entrenamiento contienen 600 imagenes y etiquetas, y la parte de test contienen 100 imagenes y etiquetas

Como vemos las X que son las encargadas del entrenamiento y de test, ambas contienen la informacion de todas las imagenes sin las etiquetas, usare la funcion reshape, para que conserven su formato de array

In [31]:
```
x_train = x_train.reshape(600,64,64)
x_test = x_test.reshape(100,64,64)

print("datos de entrenamiento: {}".format(x_train.shape))
print("datos de test: {}".format(x_test.shape))
```

```
datos de entrenamiento: (600, 64, 64)
datos de test: (100, 64, 64)
```

Como vemos tiene la division correcta y el tamaño de las imagen

Como podemos ver tiene el tamaño que se nos pide, y con la libreria de keras, pasamos los vectores a matrices binarias, esto nos ayudara en el siguiente paso que es la red neuronal.

In [32]:
```
from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

print("forma de y_train {} y el valor de y_train[0] es {}".format(y_train.shape,y_train[0]))
print("forma de y_test es {} y el valor de y_test[0] is {}".format(y_test.shape, y_test[1]))
```

```
forma de y_train (600, 2) y el valor de y_train[0] es [0. 1.]
forma de y_test es (100, 2) y el valor de y_test[0] is [1. 0.]
```

Como podemos ver, la matriz que obtenemos en y_train e y_test es una matriz binaria debido a la funcion de keras

6. Implementa dos redes neuronales convolucionales (CNN) con distintas arquitecturas siguiendo las instrucciones siguientes:*

- El número de capas convolucionales no debe ser superior a 6.

- Deben incluirse capas de agrupación(pool layers) para reducir el número de parámetros.

- En la parte inferior de la red, las capas totalmente conectadas tendrán 128 y 32 nodos, respectiva mente.

- Capa de salida con activación 'sigmoide'.*

- Los parámetros entrenables deben ser al menos 60000.

In [36]:
```
num_classes = 2
model_1 = Sequential() ## Creamos el primer modelo
# Añadimos la 1ª capa de convolución
model_1.add(Conv2D(32, kernel_size=3, activation="relu", input_shape=(64,64,1)))# formamos la primera capa co
nvolucional,con el tamaño de las imagenes
model_1.add(MaxPooling2D((2,2), padding="same"))# Vamos a agrupar los datos
model_1.add(Conv2D(16, kernel_size=3, activation="relu"))# ponemos la 2ª capa de convolución
model_1.add(Flatten()) #Ponemos la capa de flaten para reducir la entrada
# Añadimos las capas totalmente conectadas
model_1.add(Dense(128, activation="relu")) ## Añadimos las prinmeras 128 capas
model_1.add(Dense(32, activation="relu")) ## Ponemos 32 capas
model_1.add(Dense(num_classes, activation="sigmoid"))#Ponemos capa de salida
```

Creamos la segunda red neuronal o modelo

In [37]:
```
num_classes = 2
model_2 = Sequential()## creamos el primimer modelo
model_2.add(Conv2D(32, kernel_size=3, activation="relu", input_shape=(64,64,1)))# formamos la primera capa co
nvolucional, junto con el tamaño de las imagenes
model_2.add(MaxPooling2D((2,2), padding="same"))# Vamos a agrupar los datos
# Añadimos la 2ª capa de convolución
model_2.add(Conv2D(16, kernel_size=3, activation="relu"))# Ponemos la segunda convolucion con 16 capas
model_2.add(MaxPooling2D((2,2), padding="same"))#las volvemos a agrupar
# Añadimos la 3ª capa de convolución
model_2.add(Conv2D(8, kernel_size=3, activation="relu"))# # Añadimos la 3ª capa de convolución con 8 capas
model_2.add(Flatten())
model_2.add(Dense(128, activation="relu"))
model_2.add(Dense(32, activation="relu"))
model_2.add(Dense(num_classes, activation="sigmoid"))# Añadimos capa de salida, con activacion sigmoide como
 se nos pide en el enunciaedo
```

Vamos a compilar el modeloCNN, de la primera red neuronal que hemos creado, e imprimimos el modelo

In [38]:
```
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [39]:
```
print(model_1.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 62, 62, 32)        320

 max_pooling2d_2 (MaxPooling  (None, 31, 31, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 29, 29, 16)        4624

 flatten_1 (Flatten)         (None, 13456)             0

 dense_3 (Dense)             (None, 128)               1722496

 dense_4 (Dense)             (None, 32)                4128

 dense_5 (Dense)             (None, 2)                 66

=================================================================
Total params: 1,731,634
Trainable params: 1,731,634
Non-trainable params: 0
_________________________________________________________________
None
```

Como podemos ver nuestra red neuronal compila 1,731,634, y en el enunciado nos pedia 60.000, asi que nuestro modelo compila mas datos para posteriormente ser entrenados

Compilamos la segunda red neuronal o modelo,m y una vez tengo compilados los 2 entrenare las 2 redes neruonales y hare las curva Roc y podre sacar la prediccion

In [42]:
```
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [43]:
```
print(model_2.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 62, 62, 32)        320

 max_pooling2d_3 (MaxPooling  (None, 31, 31, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 29, 29, 16)        4624

 max_pooling2d_4 (MaxPooling  (None, 15, 15, 16)       0
 2D)

 conv2d_7 (Conv2D)           (None, 13, 13, 8)         1160

 flatten_2 (Flatten)         (None, 1352)              0

 dense_6 (Dense)             (None, 128)               173184

 dense_7 (Dense)             (None, 32)                4128

 dense_8 (Dense)             (None, 2)                 66

=================================================================
Total params: 183,482
Trainable params: 183,482
Non-trainable params: 0
_________________________________________________________________
None
```

Vamos a entrenar las dos redes neuronales, empezaremos enttrenamod el primer modelo o red neuronal y seguidamente entrenaremos la segunda red neuronal

In [44]:
```
# Empezamos entrenando la primera red neuronal
n_epochs = 25
mfit_1= model_1.fit(x_train, y_train, validation_data=(x_test, y_test), batch_size=10, epochs=n_epochs)
```

```
Epoch 1/25
60/60 [==============================] - 9s 8ms/step - loss: 0.6462 - accuracy: 0.6117 - val_loss: 0.5612 - v
al_accuracy: 0.7700
Epoch 2/25
60/60 [==============================] - 0s 7ms/step - loss: 0.6096 - accuracy: 0.6650 - val_loss: 0.6452 - v
al_accuracy: 0.5900
Epoch 3/25
60/60 [==============================] - 0s 7ms/step - loss: 0.5510 - accuracy: 0.7283 - val_loss: 0.5085 - v
al_accuracy: 0.7300
Epoch 4/25
60/60 [==============================] - 0s 6ms/step - loss: 0.5210 - accuracy: 0.7483 - val_loss: 0.5613 - v
al_accuracy: 0.7000
Epoch 5/25
60/60 [==============================] - 0s 6ms/step - loss: 0.4944 - accuracy: 0.7733 - val_loss: 0.4800 - v
al_accuracy: 0.7800
Epoch 6/25
60/60 [==============================] - 0s 5ms/step - loss: 0.4390 - accuracy: 0.8083 - val_loss: 0.4160 - v
al_accuracy: 0.8300
Epoch 7/25
60/60 [==============================] - 0s 6ms/step - loss: 0.4000 - accuracy: 0.8233 - val_loss: 0.4816 - v
al_accuracy: 0.8000
Epoch 8/25
60/60 [==============================] - 0s 6ms/step - loss: 0.3889 - accuracy: 0.8350 - val_loss: 0.5141 - v
al_accuracy: 0.7400
Epoch 9/25
60/60 [==============================] - 0s 6ms/step - loss: 0.3538 - accuracy: 0.8450 - val_loss: 0.4174 - v
al_accuracy: 0.8300
Epoch 10/25
60/60 [==============================] - 0s 7ms/step - loss: 0.3031 - accuracy: 0.8767 - val_loss: 0.5679 - v
al_accuracy: 0.8400
Epoch 11/25
60/60 [==============================] - 0s 5ms/step - loss: 0.3122 - accuracy: 0.8633 - val_loss: 0.4817 - v
al_accuracy: 0.8400
Epoch 12/25
60/60 [==============================] - 0s 5ms/step - loss: 0.2976 - accuracy: 0.8900 - val_loss: 0.4023 - v
al_accuracy: 0.8600
Epoch 13/25
60/60 [==============================] - 0s 4ms/step - loss: 0.2121 - accuracy: 0.9233 - val_loss: 0.5068 - v
al_accuracy: 0.8300
Epoch 14/25
60/60 [==============================] - 0s 5ms/step - loss: 0.1750 - accuracy: 0.9283 - val_loss: 0.6625 - v
al_accuracy: 0.7700
Epoch 15/25
```

```
60/60 [==============================] - 0s 4ms/step - loss: 0.1883 - accuracy: 0.9333 - val_loss: 0.6369 - v
al_accuracy: 0.8500
Epoch 16/25
60/60 [==============================] - 0s 5ms/step - loss: 0.1266 - accuracy: 0.9517 - val_loss: 0.6574 - v
al_accuracy: 0.8400
Epoch 17/25
60/60 [==============================] - 0s 4ms/step - loss: 0.1053 - accuracy: 0.9650 - val_loss: 0.6087 - v
al_accuracy: 0.8400
Epoch 18/25
60/60 [==============================] - 0s 5ms/step - loss: 0.1047 - accuracy: 0.9517 - val_loss: 0.6985 - v
al_accuracy: 0.8200
Epoch 19/25
60/60 [==============================] - 0s 5ms/step - loss: 0.0810 - accuracy: 0.9700 - val_loss: 0.8110 - v
al_accuracy: 0.8100
Epoch 20/25
60/60 [==============================] - 0s 4ms/step - loss: 0.0656 - accuracy: 0.9783 - val_loss: 0.7178 - v
al_accuracy: 0.8600
Epoch 21/25
60/60 [==============================] - 0s 4ms/step - loss: 0.0354 - accuracy: 0.9933 - val_loss: 0.9385 - v
al_accuracy: 0.7800
Epoch 22/25
60/60 [==============================] - 0s 4ms/step - loss: 0.0226 - accuracy: 0.9917 - val_loss: 0.9306 - v
al_accuracy: 0.8300
Epoch 23/25
60/60 [==============================] - 0s 4ms/step - loss: 0.0093 - accuracy: 0.9983 - val_loss: 1.1845 - v
al_accuracy: 0.8100
Epoch 24/25
60/60 [==============================] - 0s 4ms/step - loss: 0.0362 - accuracy: 0.9867 - val_loss: 1.0874 - v
al_accuracy: 0.7900
Epoch 25/25
60/60 [==============================] - 0s 4ms/step - loss: 0.0897 - accuracy: 0.9633 - val_loss: 0.9218 - v
al_accuracy: 0.8300
```

In [45]:
```
## Entrenamos la segunda red neuronal
n_epochs = 25
mfit_2 = model_2.fit(x_train, y_train, validation_data=(x_test, y_test), batch_size=10, epochs=n_epochs)
```

```
Epoch 1/25
60/60 [==============================] - 1s 7ms/step - loss: 0.6691 - accuracy: 0.5550 - val_loss: 0.5508 - v
al_accuracy: 0.7500
Epoch 2/25
60/60 [==============================] - 0s 5ms/step - loss: 0.5710 - accuracy: 0.7250 - val_loss: 0.4928 - v
al_accuracy: 0.7800
Epoch 3/25
60/60 [==============================] - 0s 4ms/step - loss: 0.4998 - accuracy: 0.7600 - val_loss: 0.4584 - v
al_accuracy: 0.8100
Epoch 4/25
60/60 [==============================] - 0s 4ms/step - loss: 0.5250 - accuracy: 0.7617 - val_loss: 0.4596 - v
al_accuracy: 0.8300
Epoch 5/25
60/60 [==============================] - 0s 4ms/step - loss: 0.4696 - accuracy: 0.7850 - val_loss: 0.4691 - v
al_accuracy: 0.8200
Epoch 6/25
60/60 [==============================] - 0s 4ms/step - loss: 0.4368 - accuracy: 0.8167 - val_loss: 0.4172 - v
al_accuracy: 0.8100
Epoch 7/25
60/60 [==============================] - 0s 4ms/step - loss: 0.3975 - accuracy: 0.8400 - val_loss: 0.4008 - v
al_accuracy: 0.8500
Epoch 8/25
60/60 [==============================] - 0s 5ms/step - loss: 0.4398 - accuracy: 0.8183 - val_loss: 0.4524 - v
al_accuracy: 0.8200
Epoch 9/25
60/60 [==============================] - 0s 4ms/step - loss: 0.3934 - accuracy: 0.8283 - val_loss: 0.4227 - v
al_accuracy: 0.8500
Epoch 10/25
60/60 [==============================] - 0s 5ms/step - loss: 0.3972 - accuracy: 0.8417 - val_loss: 0.4117 - v
al_accuracy: 0.8200
Epoch 11/25
60/60 [==============================] - 0s 4ms/step - loss: 0.3299 - accuracy: 0.8583 - val_loss: 0.5805 - v
al_accuracy: 0.7800
Epoch 12/25
60/60 [==============================] - 0s 4ms/step - loss: 0.3207 - accuracy: 0.8667 - val_loss: 0.4104 - v
al_accuracy: 0.8400
Epoch 13/25
60/60 [==============================] - 0s 5ms/step - loss: 0.2642 - accuracy: 0.8967 - val_loss: 0.4879 - v
al_accuracy: 0.8200
Epoch 14/25
60/60 [==============================] - 0s 4ms/step - loss: 0.2423 - accuracy: 0.8983 - val_loss: 0.5029 - v
al_accuracy: 0.7800
Epoch 15/25
```

```
60/60 [==============================] - 0s 4ms/step - loss: 0.2292 - accuracy: 0.9100 - val_loss: 0.5861 - v
al_accuracy: 0.8000
Epoch 16/25
60/60 [==============================] - 0s 4ms/step - loss: 0.2045 - accuracy: 0.9333 - val_loss: 0.6261 - v
al_accuracy: 0.8600
Epoch 17/25
60/60 [==============================] - 0s 5ms/step - loss: 0.1661 - accuracy: 0.9417 - val_loss: 0.6910 - v
al_accuracy: 0.8600
Epoch 18/25
60/60 [==============================] - 0s 4ms/step - loss: 0.1427 - accuracy: 0.9433 - val_loss: 0.6651 - v
al_accuracy: 0.8300
Epoch 19/25
60/60 [==============================] - 0s 4ms/step - loss: 0.1289 - accuracy: 0.9533 - val_loss: 0.8090 - v
al_accuracy: 0.8400
Epoch 20/25
60/60 [==============================] - 0s 4ms/step - loss: 0.1526 - accuracy: 0.9417 - val_loss: 0.6510 - v
al_accuracy: 0.8600
Epoch 21/25
60/60 [==============================] - 0s 4ms/step - loss: 0.1275 - accuracy: 0.9467 - val_loss: 0.8763 - v
al_accuracy: 0.8100
Epoch 22/25
60/60 [==============================] - 0s 4ms/step - loss: 0.0954 - accuracy: 0.9650 - val_loss: 0.9162 - v
al_accuracy: 0.8400
Epoch 23/25
60/60 [==============================] - 0s 4ms/step - loss: 0.1079 - accuracy: 0.9550 - val_loss: 0.8502 - v
al_accuracy: 0.8000
Epoch 24/25
60/60 [==============================] - 0s 4ms/step - loss: 0.0596 - accuracy: 0.9833 - val_loss: 1.0537 - v
al_accuracy: 0.8100
Epoch 25/25
60/60 [==============================] - 0s 4ms/step - loss: 0.0496 - accuracy: 0.9800 - val_loss: 1.0707 - v
al_accuracy: 0.8200
```

7. Comparar el rendimiento de las dos redes neuronales en el conjunto test mediante sus curvas ROC y las métricas de calidad.

Empezamos haciendo una funcion con la que podremos ver las curvas ROC de ambos modelos

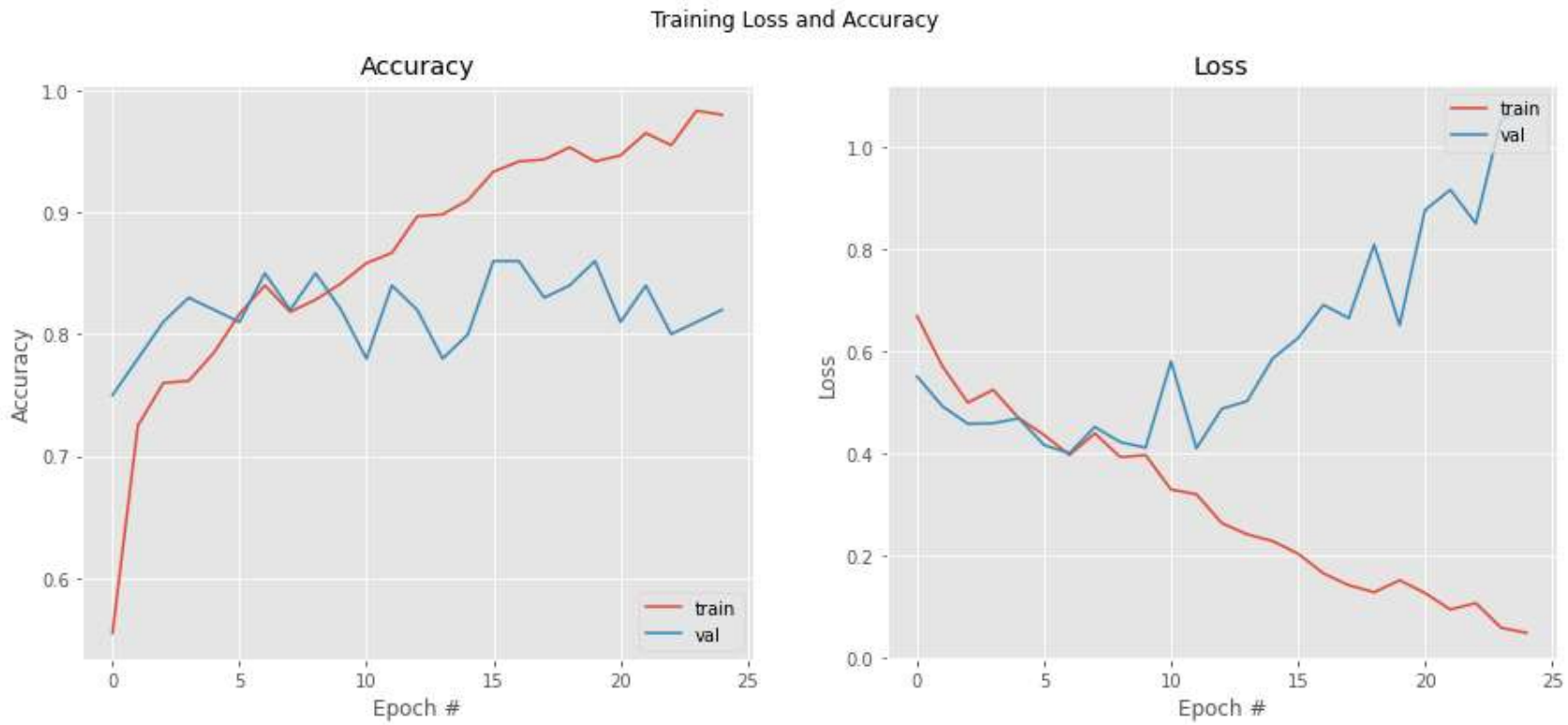
In [50]:
```
#plot accuracy and loss
def plot_prediction(n_epochs, mfit):
    N = n_epochs
    plt.style.use("ggplot")
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,6))
    fig.suptitle('Training Loss and Accuracy')

    ax1.plot(np.arange(0, N), mfit.history["accuracy"], label="train")
    ax1.plot(np.arange(0, N), mfit.history["val_accuracy"], label="val")
    ax1.set_title("Accuracy")
    ax1.set_xlabel("Epoch #")
    ax1.set_ylabel("Accuracy")
    ax1.legend(loc="lower right")

    ax2.plot(np.arange(0, N), mfit.history["loss"], label="train")
    ax2.plot(np.arange(0, N), mfit.history["val_loss"], label="val")
    ax2.set_title("Loss")
    ax2.set_xlabel("Epoch #")
    ax2.set_ylabel("Loss")
    ax2.legend(loc="upper right")

    plt.show()
```

Imprimimos el primer grafico de la primer a red neuronal

In [51]:
```
plot_prediction(n_epochs, mfit_1)
```

Imprimimos el grafifco de la segunda red neuronal

In [52]:
```
plot_prediction(n_epochs, mfit_2)
```



In [53]:
```
test_eval_1 = model_1.evaluate(x_test, y_test, verbose=1)
test_eval_2 = model_2.evaluate(x_test, y_test, verbose=1)

print('Test 1 loss:', test_eval_1[0]), print('Test 2 loss:', test_eval_2[0])
print('Test 1 accuracy:', test_eval_1[1]), print('Test 2 accuracy:', test_eval_2[1])
```

```
4/4 [==============================] - 0s 5ms/step - loss: 0.9218 - accuracy: 0.8300
4/4 [==============================] - 0s 4ms/step - loss: 1.0707 - accuracy: 0.8200
Test 1 loss: 0.9217840433120728
Test 2 loss: 1.0707391500473022
Test 1 accuracy: 0.8299999833106995
Test 2 accuracy: 0.8199999928474426
```

Out[53]: (None, None)

Con esto comparamos tanto la primera red neuronal con la segunda, como podemos ver, tiene mejor prediccion la primera CNN, ya que tiene un porcentaje de 83% de aciertos, en comparacion con la segunda que honesltamente no hay gran diferencia entre ellasque es de un 82%