Modelo de Deep Learning para la detección de COVID-19 en radiografias de Rayos-X

August 29, 2021

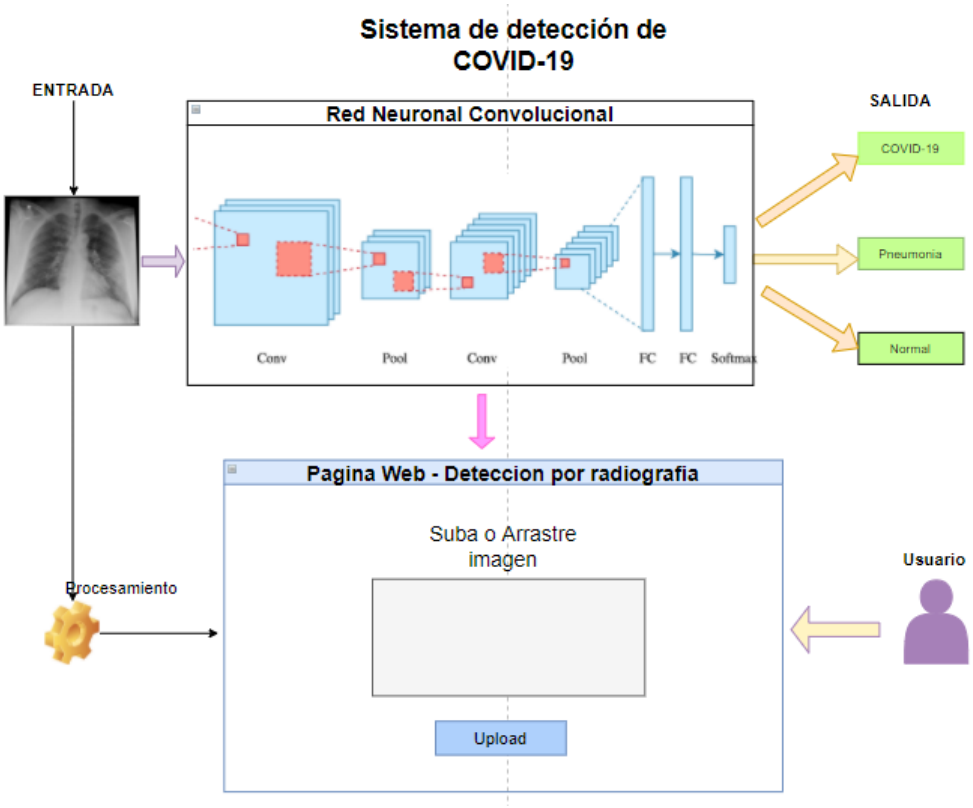
Autor: Mitma Huaccha, Johan Valerio.

1 Estado de arte

- La fuente de informacion y banco de imagenes corresponden a los creditos de los siguientes autores, reflejados en los dos siguientes articulos.
- M.E.H. Chowdhury, T. Rahman, A. Khandakar, R. Mazhar, M.A. Kadir, Z.B. Mahbub, K.R. Islam, M.S. Khan, A. Iqbal, N. Al-Emadi, M.B.I. Reaz, M. T. Islam, "Can AI help in screening Viral and COVID-19 pneumonia?" IEEE Access, Vol. 8, 2020, pp. 132665 - 132676.
- Rahman, T., Khandakar, A., Qiblawey, Y., Tahir, A., Kiranyaz, S., Kashem, S.B.A., Islam, M.T., Maadeed, S.A., Zughaier, S.M., Khan, M.S. and Chowdhury, M.E., 2020. Exploring the Effect of Image Enhancement Techniques on COVID-19 Detection using Chest X-ray Images. arXiv preprint arXiv:2012.02238.

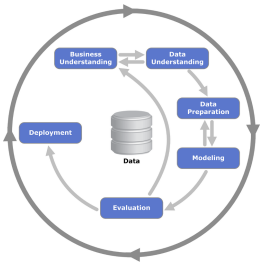
2 Metodologia CRISP-DM

- Este proyecto personal buscar brindar un modelo para detectar el COVID-19 y servir como prototipo para un futuro sistema de detección medico.
  - **Lenguaje utilizado**: Python.
  - **Librerias principales**: Scikit-Learn, Tensorflow,Keras, glob, Keras-Tuner.
  - **Deploy**: Aplicacion web.
  - **Metodologia**: Aplicacion de la metodologia de mineria de datos **CRISP-DM**.



2.0.1 COMPRENSIÓN DEL NEGOCIO

- El proposito de esta fase es alinear los objetivos de la mineria de datos con los objetivos del negocio, a fin de evitar el inicio de un proyecto que no cause ningun beneficio a la organizacion.
- **Panorama actual**: Una organizacion hospitalaria desea mejorar su precision de deteccion de covid-19 en pacientes y tener un sistema de descarte del virus en caso no cuenten con pruebas moleculares o rapidas, para poder se analizadas en el laboratorio de medicina general.
  - **Establecer los objetivos del Negocio**: Poseer un sistema que agilice el descarte inmediato de los pacientes con sospechas de **covid-19** a fin de evaluar en una segunda fase el tratamiento de dichos pacientes, y liberar al personal de atencion ante sospechas innecesarias.
  - **Evaluar la situacion actual**: El hospital no tiene un sistema de descarte inmediato de covid por radiografias, ademas en un posible escenario donde no cuenten con pruebas rapidas o moleculares, no poseen otro metodo mas eficiente para descartar dichas enfermedad.
  - **Objetivos a nivel mineria de datos**: Contar con un sistema para clasificar las radiografias de rayos-x de pacientes sospechosos de covid-19 empleando inteligencia artificial.
  - **Elaboracion de un plan de proyecto(*)** : Una vez establecidos y alineados los objetivos del data mining se procede a elaborar un proyecto que cuente con tareas,actividades y checkpoints. (*) El encargado de realizar el proyecto es un persona individual (Me).


Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment
Data

Comprension de los datos

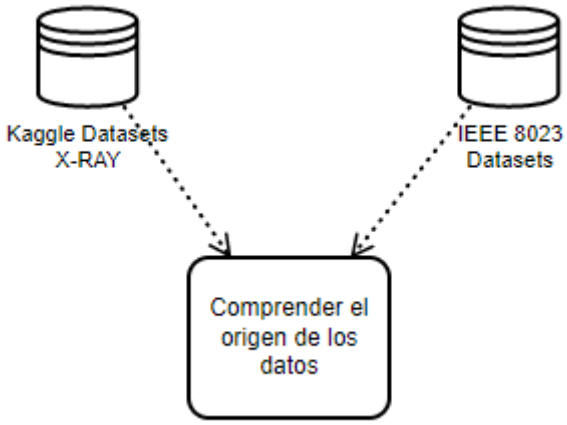
August 29, 2021

0.1 COMPRENSIÓN DE LOS DATOS

- En esta etapa de CRISP-DM esta involucrada la recopilacion de los datos y explorarlos para obtener el estado actual de los datos asi como asegurar la calidad de los datos.

Recopilacion y Origen de los datos: * Los datos se obtienen de las fuentes de **Kaggle** llamada 'Kaggle X-Ray Datasets', y el repositorio de datos en **github** 'ieee x-ray github'.

```
* Kaggle [Click Here!](https://www.kaggle.com/tawsifurrahman/covid19-radiography-database)
* Github IEEE [click Here!](https://github.com/ieee8023/covid-chestxray-dataset)
```



0.1.1 Descripcion de los datos:

- **Datos de radiografias de COVID-19 |Fuente Kaggle**: Esta fuente posee radiografias de COVID-19, Pneumonia viral, Lung Opacity y normales.
- **Datos de radiografias IEEE GITHUB**: Esta fuente posee radiografias de distintos tipos de enfermedades pulmonares. De tipos Viral, bacterial, etc. Incluyendo COVID-19.

0.1.2 Comprension de los datos:

- **Dataset de radiografias Kaggle** : Exploraremos las imagenes mas minuciosamente

[1]:
```
#exploramos la estructura de la carpeta del dataset

path_kaggle="../Datasets/KAGGLE_COVID-19_Radiography_Dataset/"
```

```
import os
for file in os.listdir(path_kaggle):
    print(file)
```

Podemos observar que la fuente de datos kaggle tiene una metadata (datos sobre los datos) por cada carpeta de imagenes asociada, asi que exploraremos dichos datos y metadata de cada uno.

Inspeccionamos los datos de COVID-19

[2]:
```
path_image_covid=os.path.join(path_kaggle,"COVID")
#contamos cuantas imagenes posee
cantidad_img=len(os.listdir(path_image_covid))
print(f"{cantidad_img} Imagenes de covid-19")
```

Mostramos algunas imagenes referentes al COVID-19

[3]:
```
#realizamos algunas importaciones de datos
import matplotlib.pyplot as plt
import random
import seaborn as sns; sns.set()
#mostramos las 9 primeras imagenes en grillas de 3x3
#desarrollamos una funcion para automatizar la muestra de imagenes por␣
 ↪condicion de entrada

def show_xrays(path=None,size_img=(3,3),target=None):
    fig,axes=plt.subplots(size_img[0],size_img[1])
    fig.set_size_inches(10,10)
    img_list=os.listdir(path)
    img_list=random.sample(img_list,size_img[0]*size_img[1])
    #plt.axis(False)
    for i,ax in enumerate(axes.flat):
        image=plt.imread(os.path.join(path,img_list[i]))
        ax.imshow(image,cmap="bone",interpolation="nearest")
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_xlabel(f"Imagen {target} -{i+1}")
    plt.show()

show_xrays(path_image_covid,target="COVID19")
```


Imagen COVID19 -1

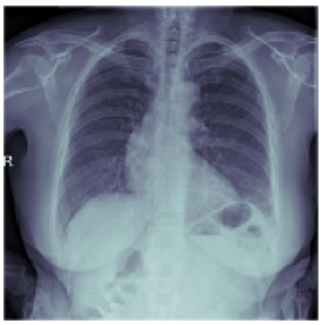


Imagen COVID19 -2


Imagen COVID19 -3

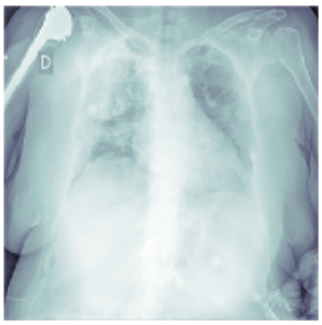


Imagen COVID19 -4

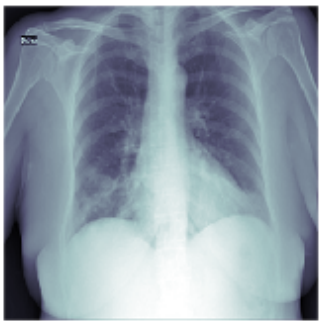


Imagen COVID19 -5


Imagen COVID19 -6


Imagen COVID19 -7


Imagen COVID19 -8

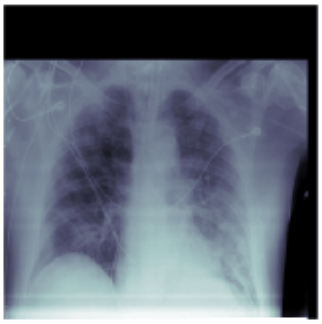


Imagen COVID19 -9

Identificaremos las fuentes y caracteristicas de las que se extrajeron los datos, esto por la metadata adjunta

[4]:
```
#importamos la libreria pandas para al manejo de DataFrames
import pandas as pd

metadata_covid=pd.read_excel(os.path.join(path_kaggle,"COVID.metadata.xlsx"))
metadata_covid.head()
```

[4]:

	FILE NAME	FORMAT	SIZE	URL
0	COVID-1	PNG	256*256	https://sirm.org/category/senza-categoria/covi…
1	COVID-2	PNG	256*256	https://sirm.org/category/senza-categoria/covi…
2	COVID-3	PNG	256*256	https://sirm.org/category/senza-categoria/covi…

```
3   COVID-4    PNG  256*256  https://sirm.org/category/senza-categoria/covi…
4   COVID-5    PNG  256*256  https://sirm.org/category/senza-categoria/covi…
```

[5]:
```
metadata_covid.FORMAT.value_counts(normalize=True).apply(lambda x:f"{x*100}%")
```

[5]:
```
PNG    100.0%
Name: FORMAT, dtype: object
```

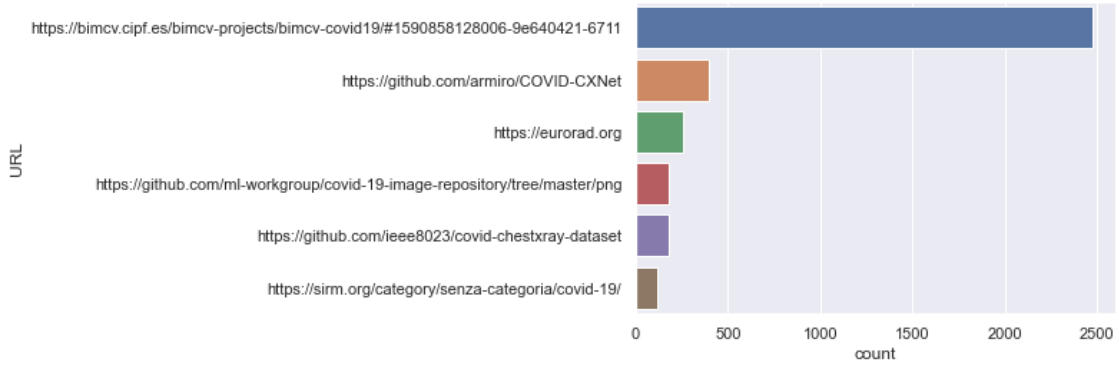
El 100% de los datos son de formato PNG y el tamaño de las imagenes son de 256*256

[6]:
```
metadata_covid.URL.value_counts(normalize=True).apply(lambda x:f"{x*100:0.
↪2f}%") #mostramos los porcentajes con 2 decimales
```

[6]:
```
https://bimcv.cipf.es/bimcv-projects/bimcv-covid19/#1590858128006-9e640421-6711
68.42%
https://github.com/armiro/COVID-CXNet
11.06%
https://eurorad.org
7.13%
https://github.com/ml-workgroup/covid-19-image-repository/tree/master/png
5.06%
https://github.com/ieee8023/covid-chestxray-dataset
5.03%
https://sirm.org/category/senza-categoria/covid-19/
3.29%
Name: URL, dtype: object
```

De aqui concluimos que el 68% de obtuvo de BIMCV, 11.06% de COVID-CXNet, entre otros.Mostramos una grafica de barras para entender mejor los datos

[7]:
```
sns.countplot(y="URL",data=metadata_covid,order=metadata_covid.URL.
↪value_counts().index)
plt.show()
```



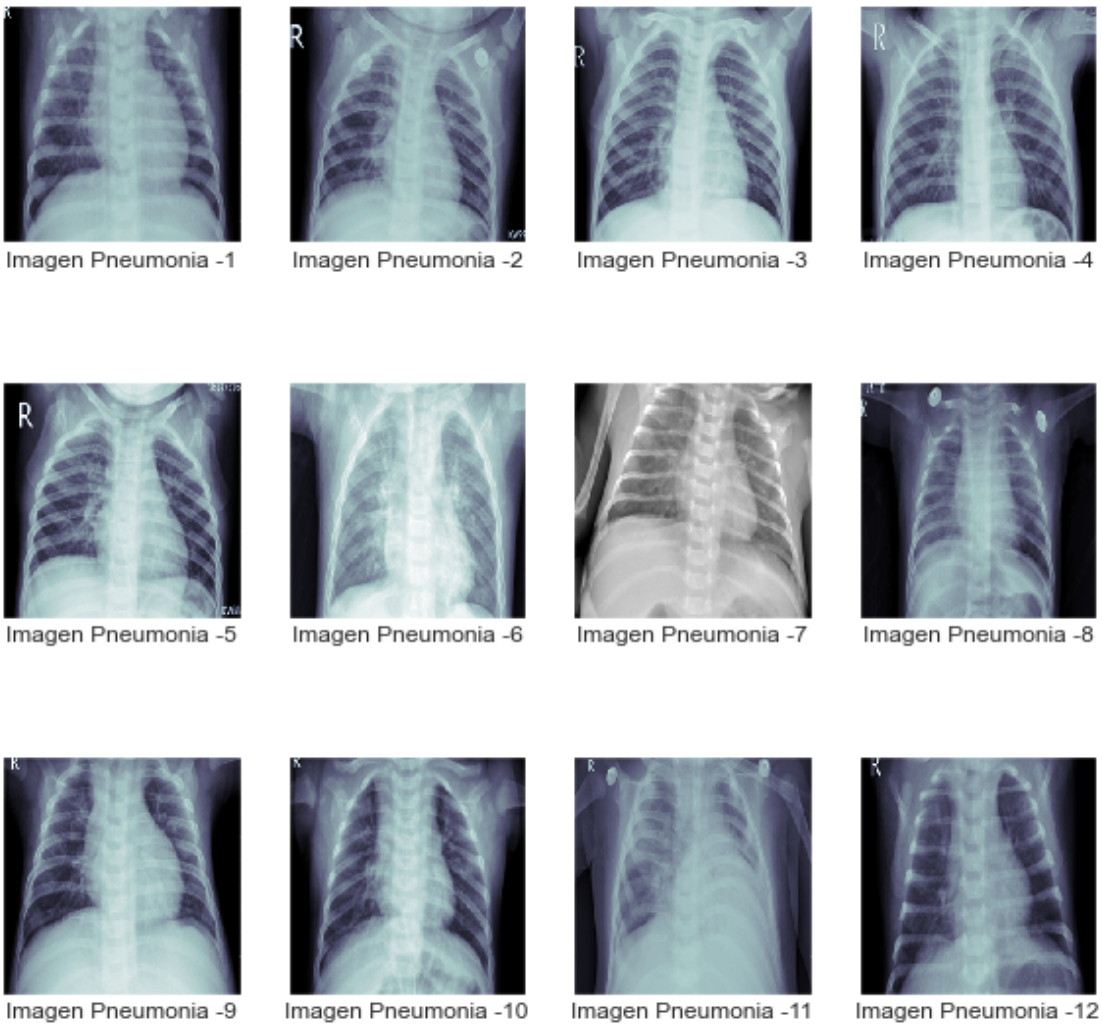
Inspeccionamos los datos Pneumonia Viral

[8]:
```
path_image_pneumonia=os.path.join(path_kaggle,"Viral Pneumonia")
#contamos cuantas imagenes posee
cantidad_img=len(os.listdir(path_image_pneumonia))
print(f"{cantidad_img} Imagenes de Pneumonia")
```

```
1345 Imagenes de Pneumonia
```

Visualizamos las radiografias de la pneumonia

[9]:
```
show_xrays(path_image_pneumonia,size_img=(3,4),target="Pneumonia")
```



[10]:
```
#mostramos y extraemos informacion de la metadata adjunta de Pneumonia 'Viral␣
↪Pneumonia.metadata.xlsx'

metadata_pneumonia=pd.read_excel(os.path.join(path_kaggle,"Viral Pneumonia.
↪metadata.xlsx"))
```

```
metadata_pneumonia.head(3) #mostramos los primeros 3 registros
```

[10]:
```
            FILE NAME FORMAT     SIZE  \
0  Viral Pneumonia-1    PNG  256*256
1  Viral Pneumonia-2    PNG  256*256
2  Viral Pneumonia-3    PNG  256*256

                                                 URL
0  https://www.kaggle.com/paultimothymooney/chest…
1  https://www.kaggle.com/paultimothymooney/chest…
2  https://www.kaggle.com/paultimothymooney/chest…
```

[11]:
```
#proporcion de formato de imagenes
metadata_pneumonia.FORMAT.value_counts(normalize=True).apply(lambda x: f"{x*100:
↪0.2f}%")
```

[11]:
```
PNG    100.00%
Name: FORMAT, dtype: object
```

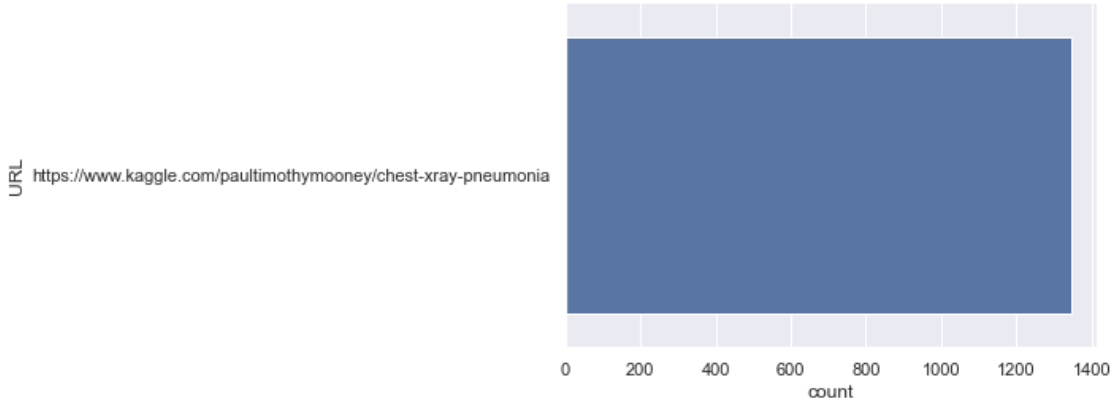
Mostramos la proporcion de la fuente de imagenes

[12]:
```
metadata_pneumonia.URL.value_counts(normalize=True).apply(lambda x: f"{x*100:0.
↪2f}%")
```

[12]:
```
https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia    100.00%
Name: URL, dtype: object
```

Interesante, la toda proporcion de imagenes de donde se obtuvo datos de pneumonia son del sitio: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

[13]:
```
#mostramos la grafica de barras para entender mejor el contexto
sns.countplot(y="URL",data=metadata_pneumonia,order=metadata_pneumonia.URL.
↪value_counts().index)
plt.show()
```



Inspeccionamos los datos Lung Opacity

- La opacificación pulmonar representa el resultado de una disminución en la proporción de gases a tejidos blandos (sangre, parénquima pulmonar y estroma) en el pulmón. Al revisar un área de mayor atenuación (opacificación) en una radiografía de tórax o TC, es vital determinar dónde está la opacificación. Los patrones se pueden dividir ampliamente en opacificación del espacio aéreo, líneas y puntos.

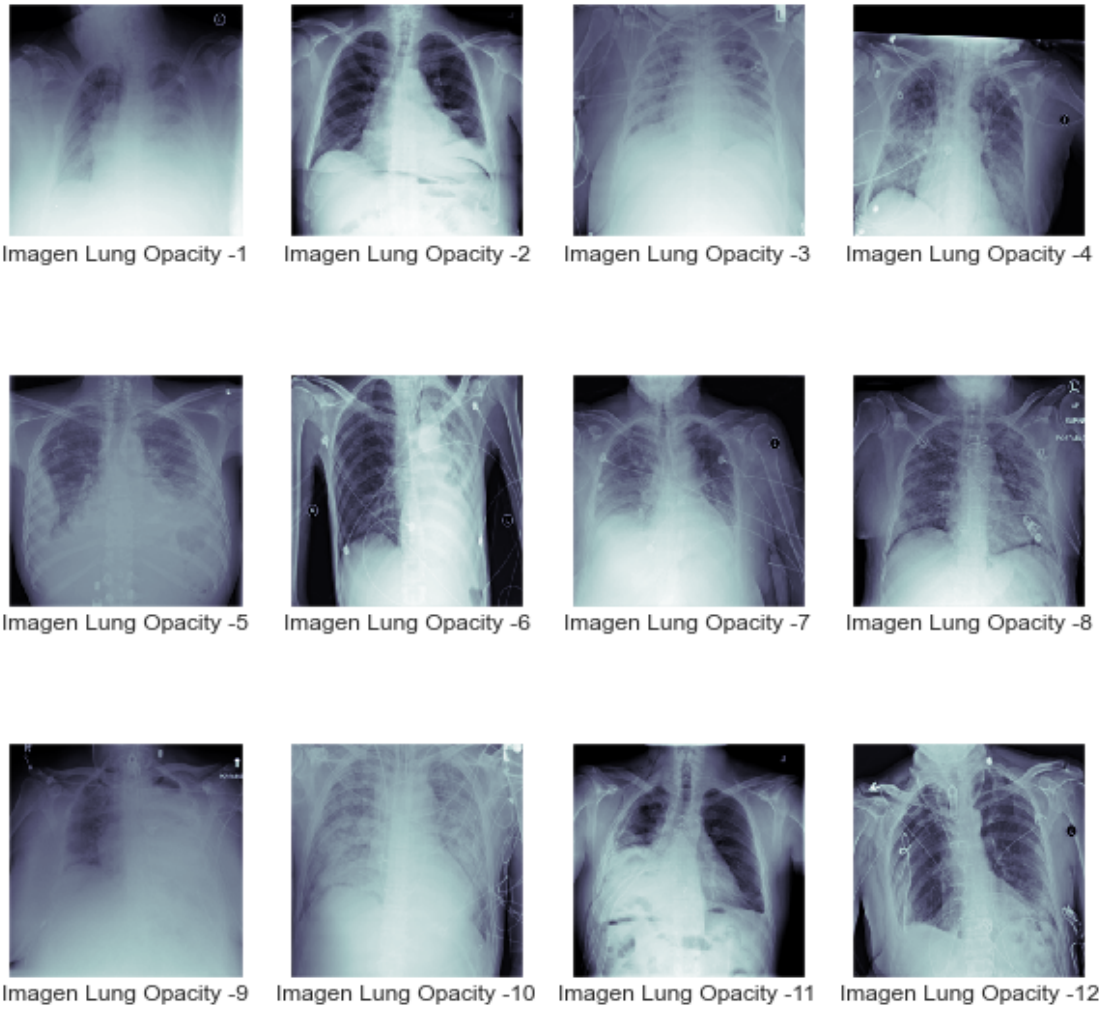
Primero mostramos la cantidad de datos

[14]:
```
path_image_lungopacity=os.path.join(path_kaggle,"Lung_Opacity")
#contamos cuantas imagenes posee
cantidad_img=len(os.listdir(path_image_lungopacity))
print(f"{cantidad_img} Imagenes de Lung_Opacity")
```

```
6012 Imagenes de Lung_Opacity
```

Mostramos las muestras de radiografias de Lung Opacity

[15]:
```
show_xrays(path_image_lungopacity,size_img=(3,4),target="Lung Opacity")
```



[16]:
```
#mostramos y extraemos informacion de la metadata adjunta de Lung opacity␣
↪'Lung_Opacity.metadata.xlsx'

metadata_lungopacity=pd.read_excel(os.path.join(path_kaggle,"Lung_Opacity.
↪metadata.xlsx"))
metadata_lungopacity.head(3) #mostramos los primeros 3 registros
```

[16]:
```
         FILE NAME FORMAT     SIZE  \
0  Lung_Opacity-1    PNG  256*256
1  Lung_Opacity-2    PNG  256*256
2  Lung_Opacity-3    PNG  256*256

                                                 URL
0  https://www.kaggle.com/c/rsna-pneumonia-detect…
1  https://www.kaggle.com/c/rsna-pneumonia-detect…
2  https://www.kaggle.com/c/rsna-pneumonia-detect…
```

[17]:
```
#proporcion de formato de imagenes
metadata_lungopacity.FORMAT.value_counts(normalize=True).apply(lambda x:␣
↪f"{x*100:0.2f}%")
```

[17]:
```
PNG    100.00%
Name: FORMAT, dtype: object
```

Mostramos la proporcion de fuentes donde se extrayeron estos datasets.

[18]:
```
metadata_lungopacity.URL.value_counts(normalize=True).apply(lambda x: f"{x*100:
↪0.2f}%")
```

[18]:
```
https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data    100.00%
Name: URL, dtype: object
```

Por lo visto los datos de Lung Opacity se obtuvieron de un repositorio de challenge de obtencion de radiografias

[19]:
```
#mostramos la grafica de barras para entender mejor el contexto
sns.countplot(y="URL",data=metadata_lungopacity,order=metadata_lungopacity.URL.
↪value_counts().index)
plt.show()
```

Inspeccionamos los datos de radiografias Normales Son aquellas que corresponden a pacientes sin ninguna afeccion pulmonar

[20]:
```
path_image_normal=os.path.join(path_kaggle,"Normal")
#contamos cuantas imagenes posee
cantidad_img=len(os.listdir(path_image_normal))
print(f"{cantidad_img} Imagenes de Normal")
```

```
10192 Imagenes de Normal
```

[21]:
```
show_xrays(path_image_normal,size_img=(3,4),target="Normal")
```


Imagen Normal -1


Imagen Normal -2


Imagen Normal -3


Imagen Normal -4


Imagen Normal -5


Imagen Normal -6


Imagen Normal -7


Imagen Normal -8

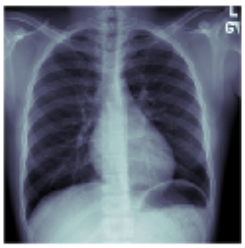


Imagen Normal -9


Imagen Normal -10


Imagen Normal -11


Imagen Normal -12

[22]:
```
#mostramos y extraemos informacion de la metadata adjunta de Normal 'Normal.
↪metadata.xlsx'

metadata_normal=pd.read_excel(os.path.join(path_kaggle,"Normal.metadata.xlsx"))
metadata_normal.head(3) #mostramos los primeros 3 registros
```

[22]:

	FILE NAME	FORMAT	SIZE	URL
0	NORMAL-1	PNG	256*256	https://www.kaggle.com/c/rsna-pneumonia-detect…
1	NORMAL-2	PNG	256*256	https://www.kaggle.com/c/rsna-pneumonia-detect…
2	NORMAL-3	PNG	256*256	https://www.kaggle.com/c/rsna-pneumonia-detect…

[23]:
```
#proporcion de formato de imagenes
metadata_normal.FORMAT.value_counts(normalize=True).apply(lambda x: f"{x*100:0.
↪2f}%")
```

[23]:
```
PNG     100.00%
Name: FORMAT, dtype: object
```

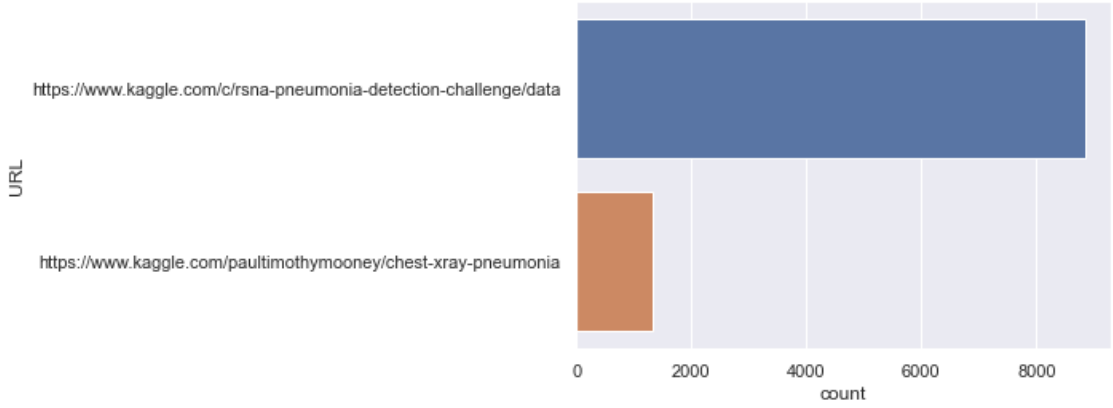
Mostramos la proporcion de fuentes de donde se extrayeron las imagenes de radiografias normales

[24]:
```
metadata_normal.URL.value_counts(normalize=True).apply(lambda x: f"{x*100:0.
↪2f}%")
```

[24]:
```
https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data    86.84%
https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia       13.16%
Name: URL, dtype: object
```

La fuentes de datos de radiografias normales son en su mayoria de **RSNA Pneumonia detection** con 86.84% de los datos, mientras que **Chest Xray Pneumonia** ocupa un 13.16% de los datos.

[25]:
```
#mostramos la grafica de barras para entender mejor el contexto
sns.countplot(y="URL",data=metadata_normal,order=metadata_normal.URL.
↪value_counts().index)
plt.show()
```



Vista general de los datos La proporcion final de las diferentes enfermedades pulmonares

[26]:
```
path_images={"COVID":0,"NORMAL":0,"Lung_Opacity":0,"Viral Pneumonia":0}
for path in path_images:
    count=len(os.listdir(os.path.join(path_kaggle,path)))
    path_images[path]=count
```

[27]:
```
for name,value in path_images.items():
    print(f"{name}: {value}")
```

```
COVID: 3616
NORMAL: 10192
Lung_Opacity: 6012
Viral Pneumonia: 1345
```

[28]:
```
for name,value in path_images.items():
    print(f"{name}: {value*100/sum(path_images.values()):0.2f}%")
```

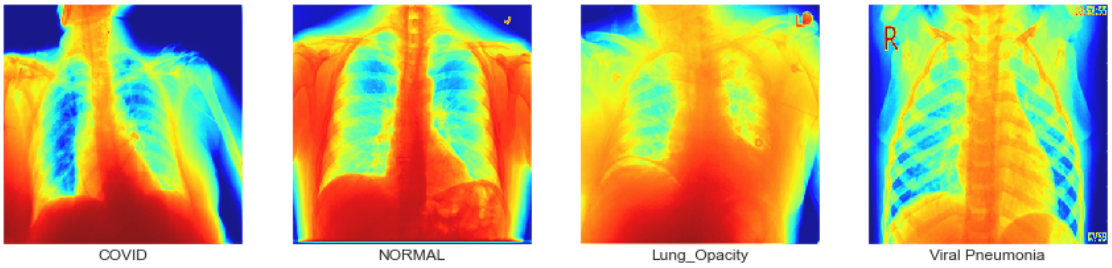
```
COVID: 17.08%
NORMAL: 48.15%
Lung_Opacity: 28.41%
Viral Pneumonia: 6.35%
```

La mayor cantidad de daros esta concentrado en radiografias Normales, seguido de LungOpacity, luego COVID, y por ultimo pneumonia viral.

Ahora realizamos una comparativa de las imagenes

[5]:
```
def select_n_image(path,n=1):
    list_img=os.listdir(path)
    img_select=random.sample(list_img,n)
    return os.path.join(path,img_select[0])
```

[30]:
```
fig,axes=plt.subplots(1,4)
fig.set_size_inches(15,4)
imagenes_elegidas=[(path,select_n_image(os.path.join(path_kaggle,path))) for␣
 ↪path in path_images.keys()] #obtnemos las rutas de la imagenes elegidas
for i,ax in enumerate(axes.flat):
    img=plt.imread(imagenes_elegidas[i][1])
    ax.imshow(img,interpolation="nearest",cmap="jet",alpha=0.9)
    ax.set_yticks([])
    ax.set_xticks([])
    ax.set_xlabel(imagenes_elegidas[i][0])
plt.show()
```



- Examinamos la estructura de los datos

[20]:
```
from PIL import Image
import os,random
import numpy as np
```

[14]:
```
path_image_select=select_n_image("../Datasets/val/Normal/")
```

[16]:
```
img=Image.open(path_image_select)
```

[24]:
```
array_img=np.array(img) #mostramos una estructura del array de la imagen
print(array_img)
```

```
[[ 0  0  0 … 82 79 75]
 [ 0  0  0 … 80 77 73]
 [ 0  0  0 … 78 76 71]
 …
 [ 0  0  0 …  0  0  1]
 [ 0  0  0 …  1  0  1]
 [ 0  0  0 …  1  1  2]]
```

[25]:
```
#miramos la dimension de los datos
print(array_img.shape)
```

```
(299, 299)
```

- **Observacion**: Las imagenes tienen un tamaño de (299x299) solo dos dimensiones lo que equivale a imagenes blanco y negro. Para el entrenamiento de la red neuronal debemos tener en cuenta dicha examinacion como dimension de entrada a la red.

Al tratarse de imagenes muy grandes, se recurre a diminuir su dimension a (256x256) en la fase de modelado haciendo uso de generadores de flujo de imagenes.

La red tendra como entrada (256,256,1): Imagenes de 256 pixeles de ancho y 256 pix largo y un canal que equivale a 1 (Blanco y negro)

Esta fase finaliza una vez coleccionado los datos y haber obtenido una vista general del problema

Preparacion de datos

August 29, 2021

1 Preparacion de datos

- Nuestra primera tarea sera la de integracion de datos.

En un principio se necesitaba integrar la data de **ieee-github** pero esta ya se encontraba integrada en la data de Kaggle, por lo cual si la usabamos ocasionaria datos duplicados o posibles problemas si esta se usaba para la validacion u otras.

- **iee-github** queda descartado por la **Fase de Compresion de los datos**

Observacion:

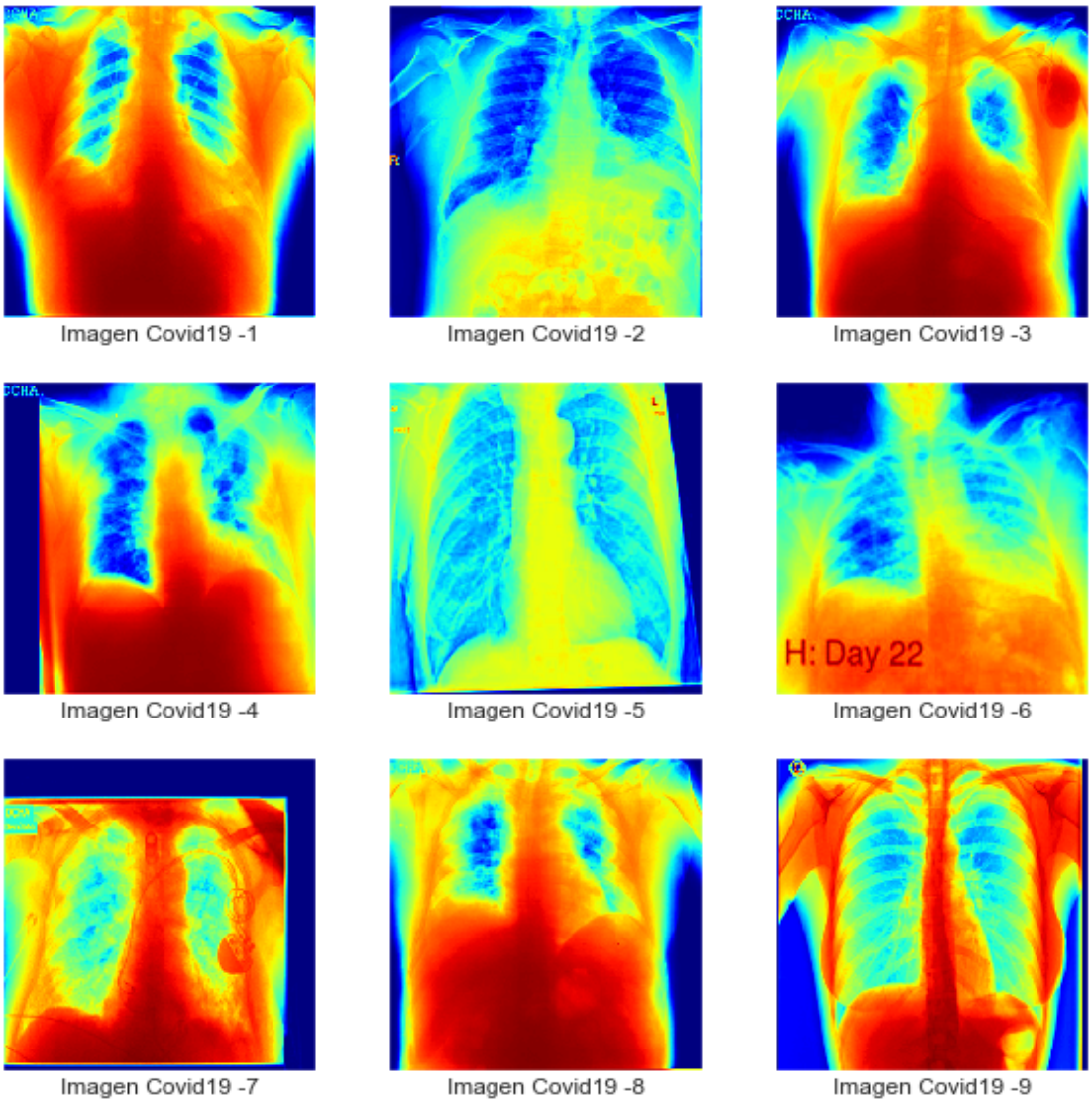
- Los datos de kaggle en su mayoria ya son datos procesados, en nuestro caso debido a que la mayoria son imagenes medicas, se estima que hayan pasado por un **Algoritmo de segmentacion de imagenes medicas** es posibles que hayan pasado por **U-Net** un algoritmo de inteligencia artificial encargado de dicha segmentacion.

Las imagenes ya estan procesadas de manera en que tienen dimensiones de 256x256 pixeles

[1]:
```
#realizamos algunas importaciones de datos
import matplotlib.pyplot as plt
import random
import seaborn as sns; sns.set()
import os
#desarrollamos una funcion para automatizar la muestra de imagenes por␣
 ↪condicion de entrada

def show_xrays(path=None,size_img=(3,3),target=None):
    fig,axes=plt.subplots(size_img[0],size_img[1])
    fig.set_size_inches(10,10)
    img_list=os.listdir(path)
    img_list=random.sample(img_list,size_img[0]*size_img[1])
    #plt.axis(False)
    for i,ax in enumerate(axes.flat):
        image=plt.imread(os.path.join(path,img_list[i]))
        ax.imshow(image,cmap="jet",interpolation="nearest")
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_xlabel(f"Imagen {target} -{i+1}")
    plt.show()
```

Visualizamos algunos datos de COVID-19.

[2]:
```
path_covid="../Datasets/KAGGLE_COVID-19_Radiography_Dataset/COVID"
show_xrays(path_covid,target="Covid19")
```



- El **escalamiento y normalizacion de datos** se usara como un flujo de **Data Augmentation** durante el entrenamiento de datos
- El **balanceo de datos** se manejara con la implementacion de una funcion de perdida mejorada para penalizar las clases mayoritarias y dar una mejor prioridad a las clases minoritarias.

1.0.1 Separación de los conjuntos de datos: Entrenamiento | Validacion | Prueba

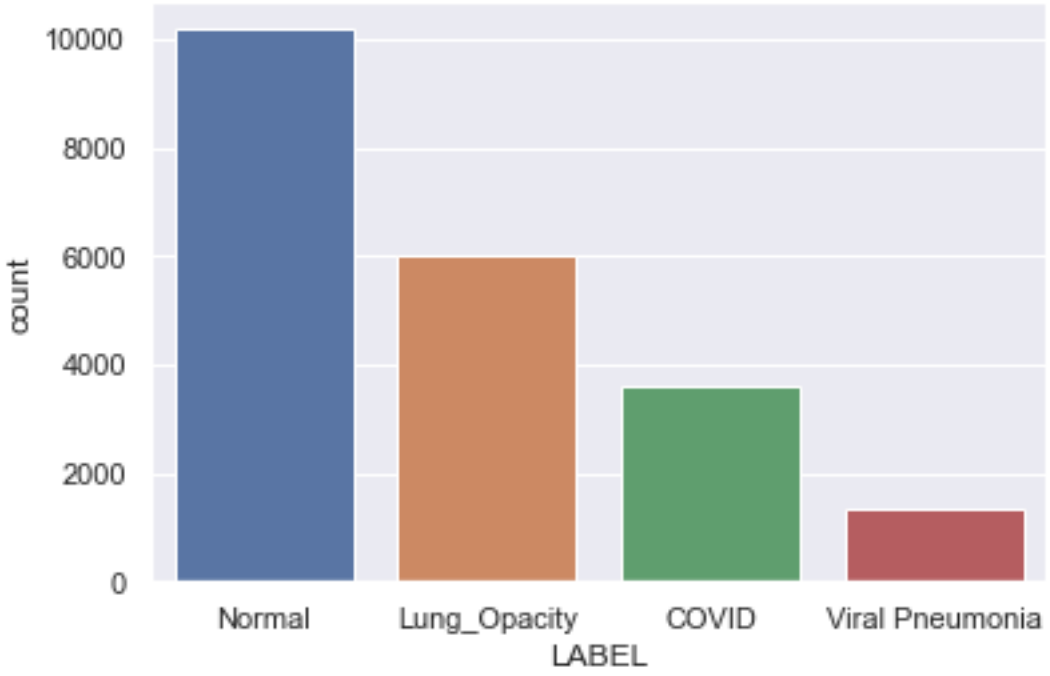
- El entrenamiento usa el 70% de los datos.

- La validacion usa el 10%.
- La prueba usa el 20%. Este conjunto no se utiliza hasta que se desarrolle el algoritmo completo y funcional, sera una contrastacion con el mundo real.

[3]:
```
#usaremos un script para automatizar el proceso de division de datos obteniendo␣
↪la ruta de cada carpeta.
import pandas as pd
def generate_dataframe_datasets():
    list_target=["COVID","Lung_Opacity","Normal","Viral Pneumonia"]
    dataset={"PATH":[],"LABEL":[]}
    for target in list_target:
        path_general=f"../Datasets/KAGGLE_COVID-19_Radiography_Dataset/{target}"
        list_img=[arch.path for arch in os.scandir(path_general) if arch.
↪is_file()]
        dataset["PATH"].extend(list_img)
        dataset["LABEL"].extend(len(list_img)*[target])

    return pd.DataFrame(dataset)
```

[30]:
```
#generar el PATH con las UBICACIONES y LABEL
data=generate_dataframe_datasets()
```

[5]:
```
sns.countplot(x="LABEL",data=data,order=data.LABEL.value_counts().index)
plt.show()
```



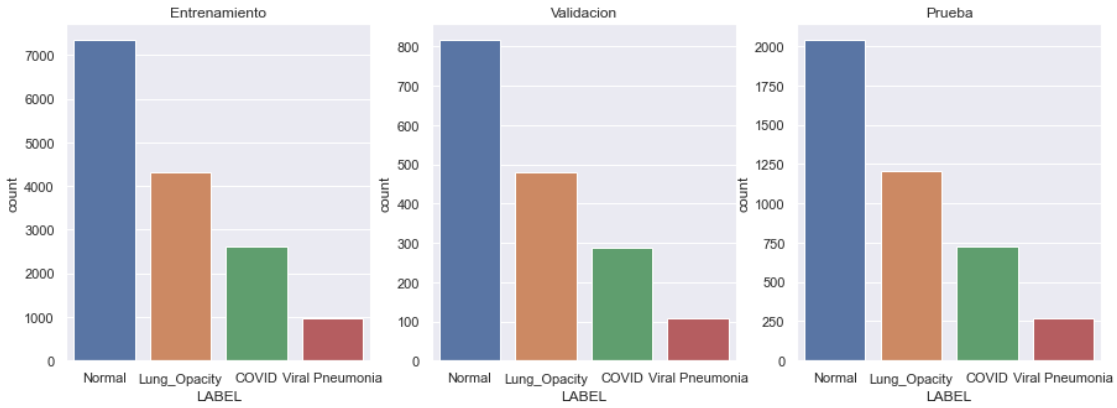
Instanciamos la clase **train__test__split** para aprovechar la funcionalidad de division **stratify**

[6]:
```
from sklearn.model_selection import train_test_split

train_data,test_data=train_test_split(data,random_state=42,test_size=0.
 ↪2,shuffle=True,stratify=data.LABEL)
train_data,val_data=train_test_split(train_data,random_state=42,test_size=0.
 ↪1,shuffle=True,stratify=train_data.LABEL)
```

[7]:
```
#script para comparar la proporcion de clases para cada particion de datos
def plot_distribution_data(data,title=None,ax=None):
    if ax is None:
        fig,(ax)=plt.subplots(1,1)
    g=sns.countplot(x="LABEL",data=data,order=data.LABEL.value_counts().
 ↪index,ax=ax)

    ax.set_title(title)
```

Mostramos como se dividieron los datos en modo grafico

[8]:
```
fig,(ax1,ax2,ax3)=plt.subplots(1,3)
fig.set_size_inches(15,5)
plot_distribution_data(train_data,title="Entrenamiento",ax=ax1)
plot_distribution_data(val_data,title="Validacion",ax=ax2)
plot_distribution_data(test_data,title="Prueba",ax=ax3)
```



Se puede apreciar que hubo una buena division estratificada de datos. Esto es muy util en los problemas de clasificacion

PROPORCION DE LOS CONJUNTOS A SEPARAR

[9]:
```
print("CONJUNTO DE ENTRENAMIENTO")
display(train_data.LABEL.value_counts(normalize=True,ascending=False).
 ↪apply(lambda x:f"{x*100:0.2f}%"))
```

```
print("CONJUNTO DE VALIDACION")
display(val_data.LABEL.value_counts(normalize=True,ascending=False).
 ↪apply(lambda x:f"{x*100:0.2f}%"))
print("CONJUNTO DE PRUEBA")
display(test_data.LABEL.value_counts(normalize=True,ascending=False).
 ↪apply(lambda x:f"{x*100:0.2f}%"))
```

```
CONJUNTO DE ENTRENAMIENTO

Normal             48.16%
Lung_Opacity       28.40%
COVID              17.09%
Viral Pneumonia     6.35%
Name: LABEL, dtype: object

CONJUNTO DE VALIDACION

Normal             48.17%
Lung_Opacity       28.39%
COVID              17.06%
Viral Pneumonia     6.38%
Name: LABEL, dtype: object

CONJUNTO DE PRUEBA

Normal             48.15%
Lung_Opacity       28.42%
COVID              17.08%
Viral Pneumonia     6.35%
Name: LABEL, dtype: object
```

NUMERO DE ELEMENTOSxCLASE DE LOS CONJUNTOS A SEPARAR

[10]:
```
print("CONJUNTO DE ENTRENAMIENTO")
display(train_data.LABEL.value_counts(ascending=False))
print("CONJUNTO DE VALIDACION")
display(val_data.LABEL.value_counts(ascending=False))
print("CONJUNTO DE PRUEBA")
display(test_data.LABEL.value_counts(ascending=False))
```

```
CONJUNTO DE ENTRENAMIENTO

Normal             7338
Lung_Opacity       4328
COVID              2604
Viral Pneumonia     968
Name: LABEL, dtype: int64

CONJUNTO DE VALIDACION

Normal             816
Lung_Opacity       481
COVID              289
```

```
Viral Pneumonia    108
Name: LABEL, dtype: int64
```

```
CONJUNTO DE PRUEBA
```

```
Normal             2038
Lung_Opacity       1203
COVID               723
Viral Pneumonia     269
Name: LABEL, dtype: int64
```

- Como es ideal, ahora los conjuntos que se dividiran tendran igual proporcion de clases, y eso evitara el sesgo en los datos, debido a que el modelo ahora vera la misma cantidad por igual.

Ahora desarrollamos el algoritmo de migracion de carpetas genericas a las destinadas train,val,test

[24]:
```
import shutil
train_data.reset_index(drop=True,inplace=True)
val_data.reset_index(drop=True,inplace=True)
test_data.reset_index(drop=True,inplace=True)

def split_data_by_info(data_info=list(),dest_path="."):
    for tipo,df in data_info:
        ruta=os.path.join(dest_path,tipo)
        if not os.path.exists(ruta): #si no existe la ruta, entonces la crea
            os.mkdir(ruta) #crear ruta
            print(f"Directorio {tipo} CREADO")
        #recorrer todo el dataset de informacion de rutas
        for i in range(df.LABEL.count()):
            subdir_name=df.loc[i,"LABEL"]
            subdir_path=os.path.join(ruta,subdir_name)
            if not os.path.exists(subdir_path): #si no existe entonces la crea
                os.mkdir(subdir_path)
                print(f"Subdirectorio {subdir_name} CREADO")
            ruta_img=df.loc[i,"PATH"]
            print(f"Moviendo {ruta_img} -> {subdir_path}")
            shutil.move(ruta_img,subdir_path)
```

[25]:
```
data_info=[("train",train_data),("val",val_data),("test",test_data)]
dest_path="../Datasets/"
```

[ ]:
```
split_data_by_info(data_info,dest_path)#la separacion se ejecuta correctamente.
```

- La separacion de datos esta lista. Ahora nos vamos a la fase de modelado

Preparacion de los datos V2

August 29, 2021

- En anteriores informes, hemos visto el bajo rendimiento que poseen los algoritmos de deep learning, lamentablemente probamos multiples configuraciones sobre los avances y diferentes implementaciones de canalizaciones sobre ellos, pero ninguno dio los resultados como yo esperaba, tener un modelo que supere el 99.99%.

A continuación mostraremos los resultados **SEMIFINALES** de los modelos seleccionados.

N°	Algoritmo	Equilibrio de datos	Epocas	Train-Accuracy	Train-Recall	Validation-Accuracy	Validation-Recall
1	Transfer-Learning sobre MobileNetV2	No balanceado	28	85.66%	83.22%	85.12%	85.52%
2	Transfer-Learning sobre MobileNetV2	Balanceado usando pon-deracion de clases	21	83.13%	79.6%	82.64%	78.63%
3	ConvNet construida	Balanceado usando Pon-deracion de clases	54	87%	86%	86%	85%
4	ConvNet construida	Balanceado usando Focal-Loss	48	89.0%	87.89%	87%	86%
5	Transfer-Learning sobre DenseNet169	Balanceado usando Focal-Loss	44	93%	93.39%	92.98%	92.38%

- Si bien el modelo con **mayor puntaje posee 92%** de precision sobre datos de validacion, este no es suficiente, porque un margen de error del 8% puede costar millones de radiografias clasificadas incorrectamente.

0.0.1 ALTERNATIVA:

- Una de las mejores alternativas que nos permitiran obtener mas puntaje es reduciendo el numero de clases haciendo uso de la seleccion de caracteristicas. De esta manera las clases **Lung Opacity** y **Pneumonia viral** quedaran bajo una categoria llamada **Enfermedades pulmonares** que no son COVID-19 ni mucho menos radiografias normales.
- Una observacion importante a realizar es que **NO SE DEBE RECURRIR A UNA CLASIFICACION BINARIA PARA ESTE TIPO DE CASOS**, asumir que daremos SI/NO ante una radiografia de COVID19 es inadecuada ya que es muy posible que una enfermedad pulmonar que posee los mismos rasgos respecto a su manifestacion en los pulmones que el COVID19 sea clasificada como tal, cuando en verdad no lo es. Debemos tener una clase intermedia entre **Normal** y **COVID19**. Y estas seran las **enfermedades pulmonares**.

PROCEDIMIENTO

- Recolectar los datos ubicados en las carpetas de dataset en train,val,split. y almacenarlos en DataFrames

[1]:
```
import os
import pandas as pd
path_train=r"..\Datasets\train"
path_val=r"..\Datasets\val"
path_test=r"..\Datasets\test"


def get_dataframe_folder(path="."):
    diccionario_frame={'ruta':[],'label':[]}
    for directorio in os.listdir(path):
        path_completo=os.path.join(path,directorio)
        for path_imagen in os.listdir(path_completo):
            diccionario_frame['ruta'].append(os.path.
 ↪join(path_completo,path_imagen))
            diccionario_frame['label'].append(directorio)
    return pd.DataFrame(diccionario_frame)
```

[32]:
```
frame_train=get_dataframe_folder(path_train)
frame_train.head()
```

[32]:

	ruta	label
0	..\Datasets\train\COVID\COVID-1.png	COVID
1	..\Datasets\train\COVID\COVID-100.png	COVID
2	..\Datasets\train\COVID\COVID-1000.png	COVID
3	..\Datasets\train\COVID\COVID-1002.png	COVID
4	..\Datasets\train\COVID\COVID-1003.png	COVID

[33]:
```
frame_val=get_dataframe_folder(path_val)
frame_val.head()
```

[33]:
```
                                  ruta  label
0  ..\Datasets\val\COVID\COVID-1009.png  COVID
1  ..\Datasets\val\COVID\COVID-1010.png  COVID
2  ..\Datasets\val\COVID\COVID-1020.png  COVID
3  ..\Datasets\val\COVID\COVID-1022.png  COVID
4  ..\Datasets\val\COVID\COVID-1024.png  COVID
```

[3]:
```
frame_test=get_dataframe_folder(path_test)
frame_test.head()
```

[3]:
```
                                  ruta  label
0    ..\Datasets\test\COVID\COVID-10.png  COVID
1  ..\Datasets\test\COVID\COVID-1001.png  COVID
2  ..\Datasets\test\COVID\COVID-1008.png  COVID
3  ..\Datasets\test\COVID\COVID-1012.png  COVID
4  ..\Datasets\test\COVID\COVID-1015.png  COVID
```

- Nueva clasificacion de las clases **pneumonias** y **lung opacity**.

[35]:
```
#para los datos de entrenamiento
frame_train['label'].replace({'Lung_Opacity':'Enfermedades Pulmonares No␣
 ↪COVID19','Viral Pneumonia':'Enfermedades Pulmonares No␣
 ↪COVID19'},inplace=True)
#para los datos de validacion
frame_val['label'].replace({'Lung_Opacity':'Enfermedades Pulmonares No␣
 ↪COVID19','Viral Pneumonia':'Enfermedades Pulmonares No␣
 ↪COVID19'},inplace=True)
#LOS DE DATOS DE TEST. TODAVIA NO SE TRANSFORMAN.
```

[4]:
```
frame_test['label'].replace({'Lung_Opacity':'Enfermedades Pulmonares No␣
 ↪COVID19','Viral Pneumonia':'Enfermedades Pulmonares No␣
 ↪COVID19'},inplace=True)
```

[5]:
```
#mover los archivos de pneumonia y lung opacity a sus respectivos destinos.␣
 ↪Enfermedades pulmonares.
import shutil
def mover_enfermedades_pulmonares(dataframe,path):
    new_dir=os.path.join(path,'Enfermedades Pulmonares No COVID19')
    os.mkdir(new_dir)
    df_result=dataframe[dataframe.label=='Enfermedades Pulmonares No␣
 ↪COVID19']['ruta']
    for ruta in df_result.tolist():
        print("Moviendo:",ruta," A ",new_dir)
        shutil.move(ruta,new_dir)
```

[ ]:
```
mover_enfermedades_pulmonares(frame_train,path_train) #el movimiento de los␣
 ↪datos de entrenamiento se ejecuta correctamente
```

[ ]:
```
mover_enfermedades_pulmonares(frame_val,path_val) #el movimiento de los datos␣
↪de validacion se ejecuta correctamente
```

[ ]:
```
mover_enfermedades_pulmonares(frame_test,path_test)  #el movimiento de los␣
↪datos de validacion se ejecuta correctamente
```

- Ahora que las clases se han fusionado, nos iremos a la fase de modelado para ingresar estos datos en la arquitectura del ultimo modelo con mas puntaje y asi poder disminuir el error de prediccion en los datos.

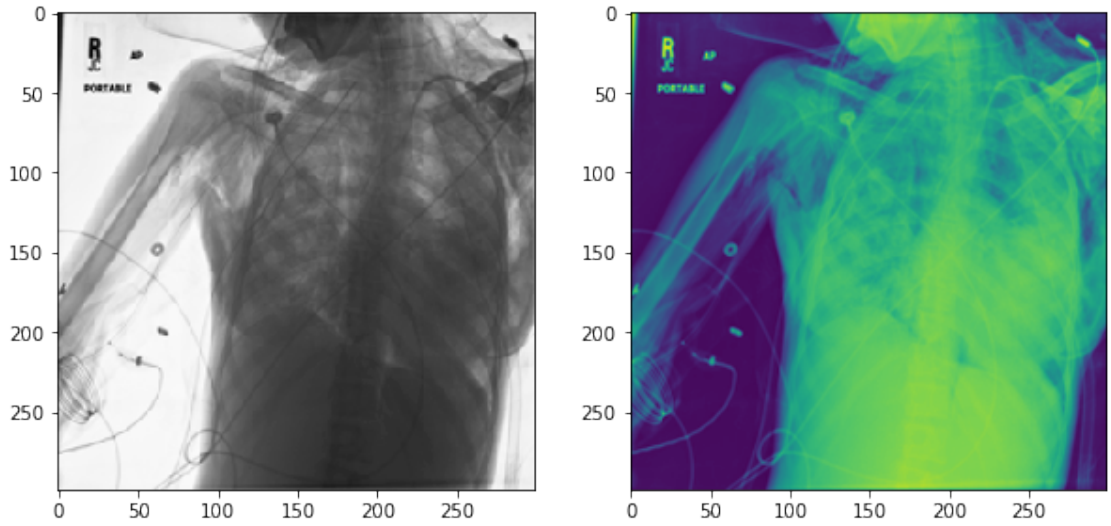
Nuevos resultados:

N°	Algoritmo (3class)	Equilibrio de datos	Epocas	Train-Accuracy	Train-Recall	Validation-Accuracy	Validation-Recall
1.B	Transfer-Learning sobre DenseNet169	Balanceo con Focal Loss	28	92%	91%	92%	91%

0.0.2 ALTERNATIVA 2: Equalizacion del SET de datos.

- Como los resultados no fueron por mucho los esperados, ahora realizamos una equalizacion de imagenes. Esto podria aumentar la calidad de las imagenes como esta:

[32]:
```
from PIL import Image,ImageOps
import matplotlib.pyplot as plt
image_test=r"..\Datasets\train\Enfermedades Pulmonares No␣
↪COVID19\Lung_Opacity-2173.png"
im=Image.open(image_test)
def compare_images(im1):
    fig,(ax1,ax2)=plt.subplots(1,2)
    fig.set_size_inches(10,10)
    ax1.imshow(im1,cmap="binary",interpolation="nearest")
    ax2.imshow(im1,cmap="viridis")
    plt.show()
compare_images(im)
```



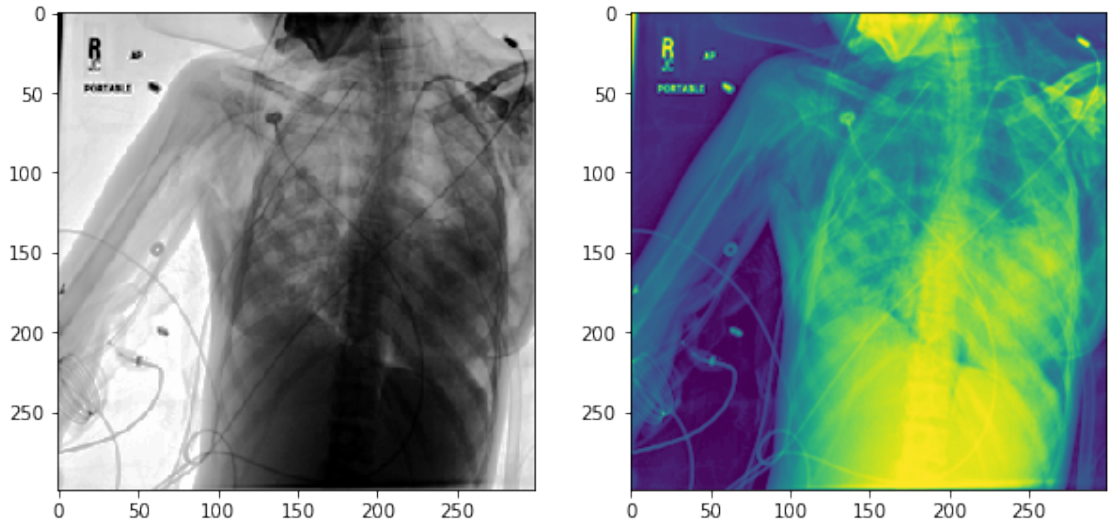
[33]:
```
display(im)
```

[34]:
```
import numpy as np

np.array(im)
```

[34]:
```
array([[171, 190, 189, …,  37,  35,  29],
       [196, 219, 216, …,  40,  36,  31],
       [201, 223, 220, …,  37,  35,  32],
       …,
       [ 29,  26,  20, …,  14,  15,  14],
       [ 19,  21,  20, …,  12,  13,  12],
       [ 10,  13,  13, …,  10,  11,  10]], dtype=uint8)
```

- A esta

[35]:
```
im_eq=ImageOps.equalize(im,mask=None)
compare_images(im_eq)
```



[36]:
```
display(im_eq)
```

[37]:
```
np.array(im_eq)
```

[37]:
```
array([[183, 228, 226, …,  63,  62,  56],
       [240, 255, 255, …,  65,  62,  58],
       [250, 255, 255, …,  63,  62,  59],
       …,
       [ 56,  53,  46, …,  36,  38,  36],
       [ 44,  47,  46, …,  27,  32,  27],
       [ 20,  32,  32, …,  20,  24,  20]], dtype=uint8)
```

0.0.3 EQUALIZACION DE IMAGENES

- Reemplazamos cada imagen con su respectiva forma ecualizada.

[50]:
```
import os

path_train=r"..\Datasets\train"
path_val=r"..\Datasets\val"
path_test=r"..\Datasets\test"
#funcion que acepta la raiz prin
def equalize(path):
    im=Image.open(path)
```

```
    im=ImageOps.equalize(im,mask=None)
    im.save(path)
    print(path,"EQUALIZADA")

def dir_equalization(root_path):
    for (root,directories,file_path) in os.walk(root_path):
        for directory in directories:
            dir_path=os.path.join(root,directory)
            for img_path in os.scandir(dir_path):
                equalize(img_path.path)
```

[ ]:
```
dir_equalization(path_train) #La ecualizacion de las imagenes de entrenamiento␣
↪se ejecuta correctamente
```

[ ]:
```
dir_equalization(path_val) #La ecualizacion de las imagenes de validacion se␣
↪ejecuta correctamente
```

[ ]:
```
dir_equalization(path_test) #La ecualizacion de las imagenes de test se ejecuta␣
↪correctamente
```

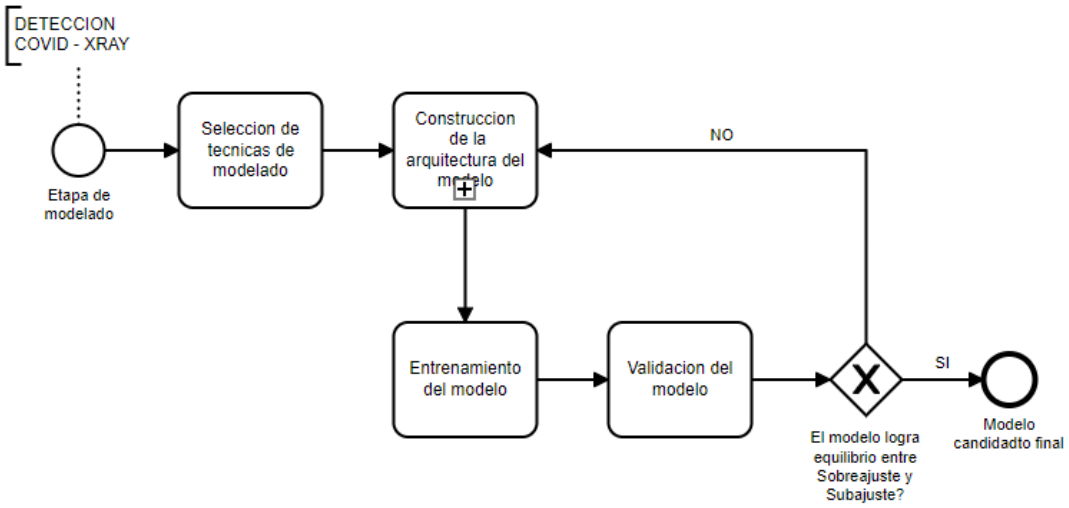
- Ahora las imagenes de entrenamiento, validacion y test estan ecualizadas. Y listas para el nuevo entrenamiento de datos.

Modelado

August 29, 2021

Esta fase de la metdologia consiste en extraer el valor de los datos desarrollando un modelo que aprenda de los patrones en los datos.

- El diagrama en cuestion de esta fase esta a continuacion:



0.0.1 Selección de tecnicas de Modelado

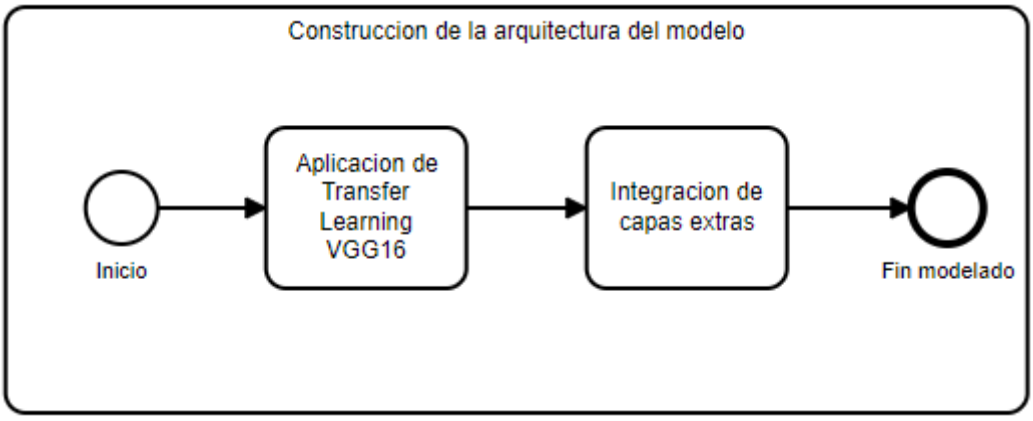
Al tratarse de un problema de clasificacion de imagenes entre los posibles candidatos tenemos:

- MultiLayer Pereptron: Red neuronal de capas densamente conectadas
- Convolutional Neuronal Network: Red neuronal convolucional.
- Modelos de machine learning clasico (Maquinas de soporte vectorial, arboles de decision e impulso, etc.)

Escogi la red neuronal convolucional porque **aprende de patrones locales** como rasgos pequeños y en bloques de informacion, mientras que el **MLP** aprende de patrones especficos, e decir de todo el espacio de entrada en general.

0.0.2 Construcción de la arquitectura del modelo

- Para la construccion de la arquitectura aplicaremos Transfer Learning de la arquitectura **MobileNETV2** y la acomplaremos anuestras capas personalizadas para el problema.



[2]:
```
import tensorflow as tf
from tensorflow.keras.utils import plot_model
```

- Implementamos la arquitectura de MobileNet V2, por ser liviana y consumir menos recursos.

[3]:
```
INPUT_SHAPE=(256,256,3) #256x256  pixeles, y 3 canales de colores(R,G,B)

model_mobilenetv2=tf.keras.applications.MobileNetV2(
                                                    input_shape=INPUT_SHAPE,
                                                    weights="imagenet", #obtener los␣
 ↪pesos del entrenamiento con las imagenes ImageNetV3
                                                    include_top=False, #no obtener el␣
 ↪clasificador final (LO PERSONALIZAMOS),
                                    )
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in
[96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as
the default.
```

0.0.3 Arquitectura del modelo adaptado

- Capa de entrada **Input** para las imagenes.
- Capa de Modelo **MobileNetv2** para hacer **Transfer Learning**.
- Capa MaxAveragePooling.
- Capa Densa personalizada que responda a las clases a predecir.

Para esta implementacion usaremos la **API Functional**.

[5]:
```
N_CLASSES=4 #Clases a predecir ->COV
BASE_LEARNING_RATE=0.0001
def build_model_base():
    model_mobilenetv2.trainable=False #las capas del modelo no actualizan su␣
 ↪peso.
    inputs=tf.keras.layers.Input(shape=INPUT_SHAPE)
    x=model_mobilenetv2(inputs)
```

```
    x=tf.keras.layers.GlobalAveragePooling2D()(x)
    outputs=tf.keras.layers.Dense(units=N_CLASSES,activation="softmax")(x)

    model=tf.keras.Model(inputs,outputs)

    #compilamos el modelo
    model.compile(loss="categorical_crossentropy",
                  optimizer=tf.keras.optimizers.Adam(lr=BASE_LEARNING_RATE),
                  metrics=["accuracy",tf.keras.metrics.Recall()]
                  )
    return model
```

[6]:
```
model_base=build_model_base()
model_base.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 256, 256, 3)]     0
_________________________________________________________________
mobilenetv2_1.00_224 (Functi (None, 8, 8, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 4)                 5124
=================================================================
Total params: 2,263,108
Trainable params: 5,124
Non-trainable params: 2,257,984
_________________________________________________________________
```

- Carga y aumento de datos

[7]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator #generator␣
↪de imagenes
```

- Configuracion del generador de datasets de entrenamiento, validacion y prueba

[8]:
```
train_datagen=ImageDataGenerator(
            rescale=1.0/127.5, #escalamos los datos en rangos de [-1,1] El␣
↪modelo espera esta configuracion
            zoom_range=0.2, #agregamos rotacion a las imagenes
            rotation_range=5,
            horizontal_flip=True
        )
#sobre los datos de validacion y test no se hace ningun aumento de datos.
```

```
validation_datagen=ImageDataGenerator(rescale=1.0/127.5) #escalamiento de␣
↪validacion a un rango de [-1,1]
test_datagen=ImageDataGenerator(rescale=1.0/127.5)       #escalamiento de test␣
↪a un rango de [-1,1]
```

- Obtencion de los datos para generar

[9]:
```
#definimos las rutas para el acceso a los datos
train_path="../Datasets/train/"
validation_path="../Datasets/val/"
test_path="../Datasets/test/"

#creamos los generadores de datos a partir de los flujos de informacion
BATCH_SIZE=32 #tamaño del lote que se ira pasando poco a poco
IMAGE_SIZE=(256,256)

train_generator=train_datagen.flow_from_directory(
    train_path,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)

validation_generator=train_datagen.flow_from_directory(
    validation_path,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)

test_generator=train_datagen.flow_from_directory(
    test_path,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)
```

```
Found 15238 images belonging to 4 classes.
Found 1694 images belonging to 4 classes.
Found 4233 images belonging to 4 classes.
```

- Generando **Callbacks** para detener el entrenamiento cuando no se tienen buenos resultados

[12]:
```
early_stopping=tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=3)␣
↪#cuando la funcion de perdida ya no mejora
n_epochs=10
```

0.0.4 Entrenamiento el modelo BASE

[10]:
```
model_base=build_model_base()
model_history=model_base.fit_generator(
                    train_generator,
                    epochs=n_epochs,
                    validation_data=validation_generator,
                    validation_steps=validation_generator.samples//BATCH_SIZE,
                    callbacks=[early_stopping]
            )
```

```
C:\Users\avira\anaconda3\envs\mlearning\lib\site-
packages\tensorflow\python\keras\engine\training.py:1844: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
```

```
Epoch 1/10
477/477 [==============================] - 839s 2s/step - loss: 1.0436 -
accuracy: 0.5648 - recall_1: 0.3902 - val_loss: 0.6909 - val_accuracy: 0.7302 -
val_recall_1: 0.6382
Epoch 2/10
477/477 [==============================] - 748s 2s/step - loss: 0.6853 -
accuracy: 0.7348 - recall_1: 0.6481 - val_loss: 0.6096 - val_accuracy: 0.7668 -
val_recall_1: 0.7067
Epoch 3/10
477/477 [==============================] - 738s 2s/step - loss: 0.6108 -
accuracy: 0.7637 - recall_1: 0.6996 - val_loss: 0.5683 - val_accuracy: 0.7794 -
val_recall_1: 0.7260
Epoch 4/10
477/477 [==============================] - 719s 2s/step - loss: 0.5620 -
accuracy: 0.7862 - recall_1: 0.7325 - val_loss: 0.5445 - val_accuracy: 0.7951 -
val_recall_1: 0.7404
Epoch 5/10
477/477 [==============================] - 734s 2s/step - loss: 0.5310 -
accuracy: 0.7994 - recall_1: 0.7484 - val_loss: 0.5321 - val_accuracy: 0.7993 -
val_recall_1: 0.7452
Epoch 6/10
477/477 [==============================] - 721s 2s/step - loss: 0.5199 -
accuracy: 0.8011 - recall_1: 0.7550 - val_loss: 0.5063 - val_accuracy: 0.8035 -
val_recall_1: 0.7578
Epoch 7/10
477/477 [==============================] - 727s 2s/step - loss: 0.4895 -
accuracy: 0.8181 - recall_1: 0.7757 - val_loss: 0.4886 - val_accuracy: 0.8173 -
val_recall_1: 0.7728
Epoch 8/10
477/477 [==============================] - 678s 1s/step - loss: 0.4913 -
accuracy: 0.8138 - recall_1: 0.7725 - val_loss: 0.4737 - val_accuracy: 0.8281 -
val_recall_1: 0.7873
```

```
Epoch 9/10
477/477 [==============================] - 687s 1s/step - loss: 0.4707 -
accuracy: 0.8206 - recall_1: 0.7840 - val_loss: 0.4557 - val_accuracy: 0.8395 -
val_recall_1: 0.7945
Epoch 10/10
477/477 [==============================] - 679s 1s/step - loss: 0.4716 -
accuracy: 0.8201 - recall_1: 0.7834 - val_loss: 0.4703 - val_accuracy: 0.8269 -
val_recall_1: 0.7861
```

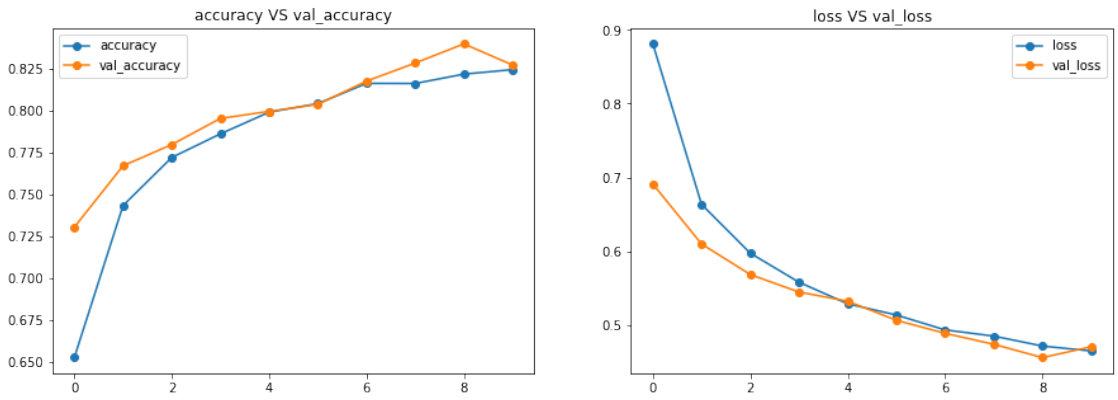
[16]:
```
#deseo guardar el History devuelto del entrenamiento
dfhist=pd.DataFrame(model_history.history)
dfhist.to_csv("model_historyV1.csv",index=False)
```

Mostramos los resultados encontrados

[20]:
```
import pandas as pd
import matplotlib.pyplot as plt
def plot_results(history,columns=[],ax=None):
    if ax is None:
        fig,(ax)=plt.subplots(1,1)
    df=pd.DataFrame(history)[columns]
    df.plot(kind="line",ax=ax,style="o-")
    ax.set_title(" VS ".join(columns))
```

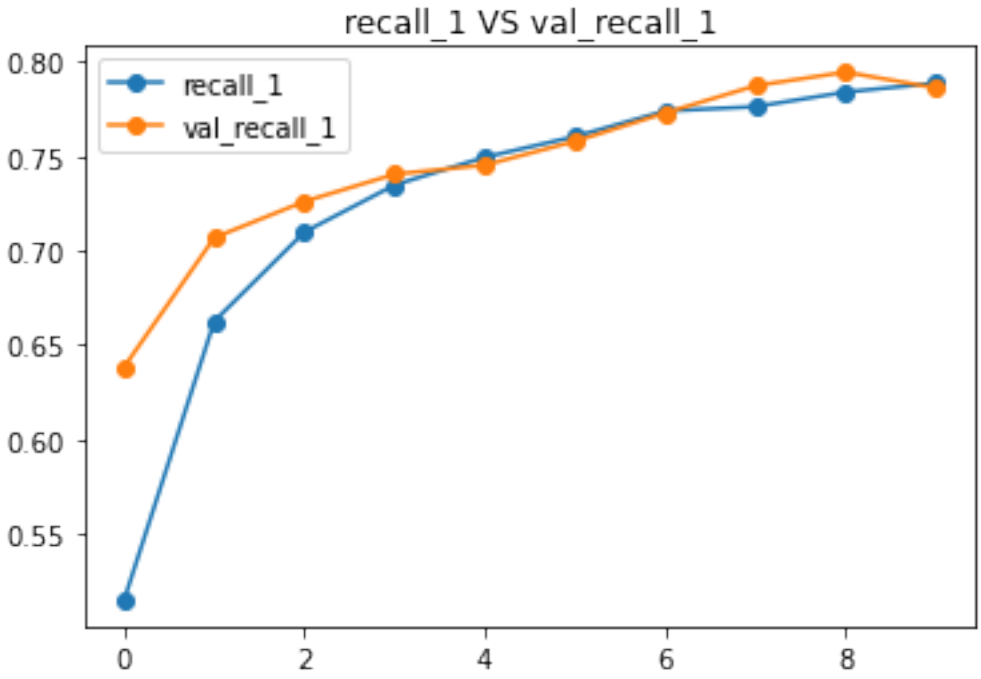
Trazamos la curva de la accuracy y funcion de perdida

[32]:
```
#la curva de precision
history=model_history.history
fig,(ax1,ax2)=plt.subplots(1,2)
fig.set_size_inches(15,5)
plot_results(history,["accuracy","val_accuracy"],ax=ax1)
plot_results(history,["loss","val_loss"],ax=ax2)
#plt.title("Accuracy & Loss function evolution")
plt.show()
```



- Para **recall**

[34]:
```
plot_results(history,["recall_1","val_recall_1"])
```



[38]:
```
#guardamos el modelo base con 10 epochs
tf.saved_model.save(model_base,"baseline")
model_base.save("baseline.h5")
```

```
INFO:tensorflow:Assets written to: baseline\assets
```

2da fase del entrenamiento (2/3)

[10]:
```
#sesion de entramiento restaurada
#El primer entramiento tuvo solo 10 epocas, por temas de tiempo
#El 2do entrenamiento retoma los datos y establece 10 epocas mas, dando lugar␣
 ↪asi a 20 epocas en total.
model_base=tf.keras.models.load_model("baseline.h5")
```

[14]:
```
n_epochs=20
model_base.fit_generator(
        train_generator,
        epochs=n_epochs,
        initial_epoch=10,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples//BATCH_SIZE,
        callbacks=[early_stopping]
```

```
    )
```

```
C:\Users\avira\anaconda3\envs\mlearning\lib\site-
packages\tensorflow\python\keras\engine\training.py:1844: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
```

```
Epoch 11/20
477/477 [==============================] - 733s 2s/step - loss: 0.4540 -
accuracy: 0.8288 - recall_1: 0.7958 - val_loss: 0.4646 - val_accuracy: 0.8317 -
val_recall_1: 0.7903
Epoch 12/20
477/477 [==============================] - 793s 2s/step - loss: 0.4459 -
accuracy: 0.8324 - recall_1: 0.8012 - val_loss: 0.4575 - val_accuracy: 0.8167 -
val_recall_1: 0.7885
Epoch 13/20
477/477 [==============================] - 823s 2s/step - loss: 0.4431 -
accuracy: 0.8353 - recall_1: 0.8025 - val_loss: 0.4471 - val_accuracy: 0.8431 -
val_recall_1: 0.8077
Epoch 14/20
477/477 [==============================] - 873s 2s/step - loss: 0.4371 -
accuracy: 0.8350 - recall_1: 0.8033 - val_loss: 0.4470 - val_accuracy: 0.8389 -
val_recall_1: 0.8035
Epoch 15/20
477/477 [==============================] - 874s 2s/step - loss: 0.4299 -
accuracy: 0.8394 - recall_1: 0.8098 - val_loss: 0.4514 - val_accuracy: 0.8269 -
val_recall_1: 0.7981
Epoch 16/20
477/477 [==============================] - 850s 2s/step - loss: 0.4267 -
accuracy: 0.8443 - recall_1: 0.8120 - val_loss: 0.4428 - val_accuracy: 0.8389 -
val_recall_1: 0.8029
Epoch 17/20
477/477 [==============================] - 845s 2s/step - loss: 0.4210 -
accuracy: 0.8450 - recall_1: 0.8155 - val_loss: 0.4238 - val_accuracy: 0.8419 -
val_recall_1: 0.8071
Epoch 18/20
477/477 [==============================] - 831s 2s/step - loss: 0.4187 -
accuracy: 0.8431 - recall_1: 0.8134 - val_loss: 0.4220 - val_accuracy: 0.8413 -
val_recall_1: 0.8203
Epoch 19/20
477/477 [==============================] - 811s 2s/step - loss: 0.4149 -
accuracy: 0.8468 - recall_1: 0.8179 - val_loss: 0.4351 - val_accuracy: 0.8444 -
val_recall_1: 0.8137
Epoch 20/20
477/477 [==============================] - 768s 2s/step - loss: 0.4078 -
accuracy: 0.8462 - recall_1: 0.8211 - val_loss: 0.4220 - val_accuracy: 0.8389 -
val_recall_1: 0.8155
```

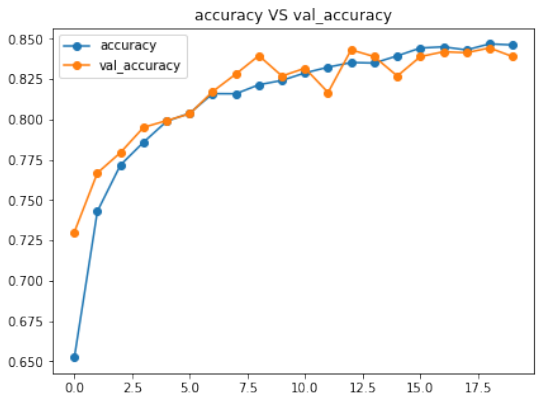
[14]:
```
<tensorflow.python.keras.callbacks.History at 0x1a329708520>
```

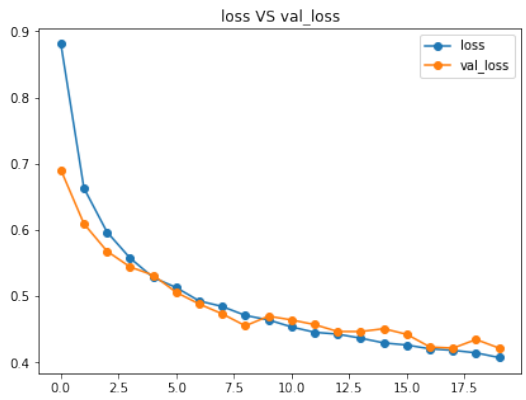
[17]:
```
import pandas as pd
#deseo guardar el History devuelto del entrenamiento
model_history=_
dfhist=pd.DataFrame(model_history.history)
dfhist.to_csv("model_historyV2.csv",index=False)
```

- Graficar las dos sesiones de entrenamiento

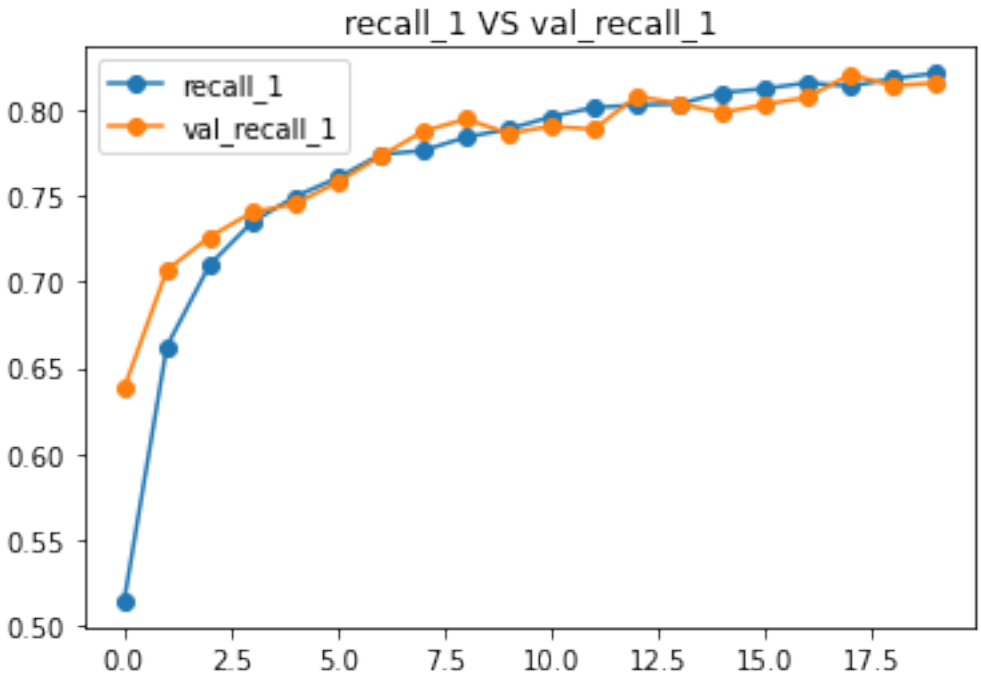
[18]:
```
df1=pd.read_csv("model_historyV1.csv",sep=",")
df2=pd.read_csv("model_historyV2.csv",sep=",")
df_history=pd.concat([df1,df2],axis=0)
```

[21]:
```
#la curva de precision
history=df_history.to_dict("list")
fig,(ax1,ax2)=plt.subplots(1,2)
fig.set_size_inches(15,5)
plot_results(history,["accuracy","val_accuracy"],ax=ax1)
plot_results(history,["loss","val_loss"],ax=ax2)
#plt.title("Accuracy & Loss function evolution")
plt.show()
```





[22]:
```
plot_results(history,["recall_1","val_recall_1"])
```



- Guardamos el modelo base final. sesion (2/3)

[23]:
```
model_base.save("baselinev2.h5")
```

- Evaluamos el puntaje en el **conjunto de entrenamiento**.

[24]:
```
model_base.evaluate(train_generator)
```

```
477/477 [==============================] - 650s 1s/step - loss: 0.4047 -
accuracy: 0.8498 - recall_1: 0.8218
```

[24]:
```
[0.40467602014541626, 0.8498490452766418, 0.8217613697052002]
```

- Evaluamos el puntaje en el **conjunto de validacion**.

[25]:
```
model_base.evaluate(validation_generator)
```

```
53/53 [==============================] - 77s 1s/step - loss: 0.4248 - accuracy:
0.8430 - recall_1: 0.8117
```

[25]:
```
[0.42475569248199463, 0.8429751992225647, 0.8116883039474487]
```

- **RESULTADOS PARCIALES**:

El accuracy para el conjunto de entrenamiento es de **ACC=84.98%** y **Recall=82.18%**.

El accuracy para el conjunto de validación es de **ACC=84.29%** y **Recall=81.16%**.

3era fase de entrenamiento (3/3)

- Retomamos el entrenamiento pasado y continuamos el entrenamiento

[28]:
```
epochs=30
early_stopping=tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=3)␣
 ↪#cuando la funcion de perdida ya no mejora
BATCH_SIZE=32
model_history=model_base.fit_generator(
                        train_generator,
                        epochs=epochs,
                        initial_epoch=20,
                        validation_data=validation_generator,
                        validation_steps=validation_generator.samples//
 ↪BATCH_SIZE,
                        callbacks=[early_stopping]
                )
```

```
C:\Users\avira\anaconda3\envs\mlearning\lib\site-
packages\tensorflow\python\keras\engine\training.py:1844: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
```

```
Epoch 21/30
477/477 [==============================] - 839s 2s/step - loss: 0.4072 -
accuracy: 0.8470 - recall_1: 0.8196 - val_loss: 0.4243 - val_accuracy: 0.8431 -
val_recall_1: 0.8083
Epoch 22/30
477/477 [==============================] - 866s 2s/step - loss: 0.4028 -
accuracy: 0.8521 - recall_1: 0.8267 - val_loss: 0.4133 - val_accuracy: 0.8450 -
val_recall_1: 0.8203
Epoch 23/30
477/477 [==============================] - 849s 2s/step - loss: 0.4043 -
accuracy: 0.8512 - recall_1: 0.8255 - val_loss: 0.4106 - val_accuracy: 0.8438 -
val_recall_1: 0.8191
Epoch 24/30
477/477 [==============================] - 826s 2s/step - loss: 0.4070 -
accuracy: 0.8495 - recall_1: 0.8236 - val_loss: 0.4160 - val_accuracy: 0.8534 -
val_recall_1: 0.8233
Epoch 25/30
477/477 [==============================] - 817s 2s/step - loss: 0.3970 -
accuracy: 0.8502 - recall_1: 0.8248 - val_loss: 0.4008 - val_accuracy: 0.8468 -
val_recall_1: 0.8263
Epoch 26/30
477/477 [==============================] - 904s 2s/step - loss: 0.3924 -
accuracy: 0.8508 - recall_1: 0.8271 - val_loss: 0.4045 - val_accuracy: 0.8564 -
val_recall_1: 0.8299
Epoch 27/30
```

```
477/477 [==============================] - 854s 2s/step - loss: 0.3935 -
accuracy: 0.8548 - recall_1: 0.8299 - val_loss: 0.4068 - val_accuracy: 0.8462 -
val_recall_1: 0.8239
Epoch 28/30
477/477 [==============================] - 847s 2s/step - loss: 0.3913 -
accuracy: 0.8569 - recall_1: 0.8325 - val_loss: 0.4144 - val_accuracy: 0.8486 -
val_recall_1: 0.8251
```

[29]:
```
dtf=pd.DataFrame(model_history.history)
dtf.to_csv("model_historyV3.csv",index=False)
```

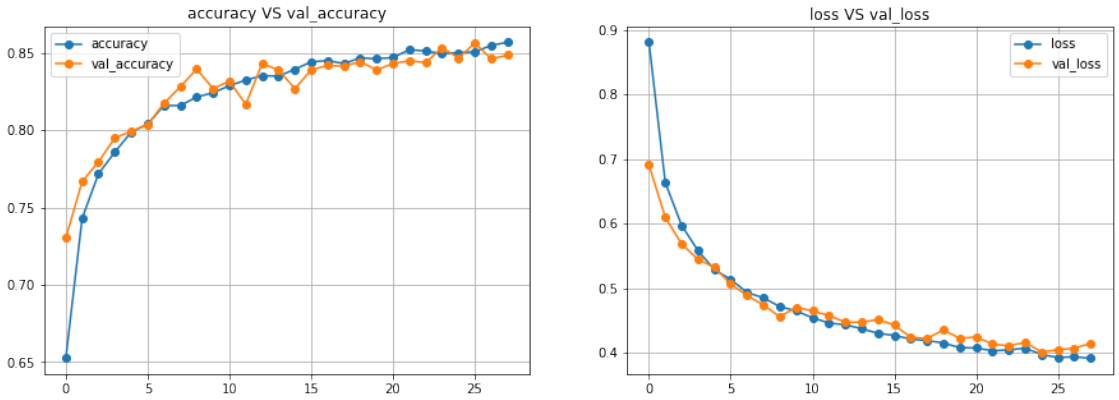
- Volvemos a ejecutar la secuencia de pasos de verificacion de resultados

[30]:
```
df1=pd.read_csv("model_historyV1.csv",sep=",")
df2=pd.read_csv("model_historyV2.csv",sep=",")
df3=pd.read_csv("model_historyV3.csv",sep=",")
df_history=pd.concat([df1,df2,df3],axis=0)
```

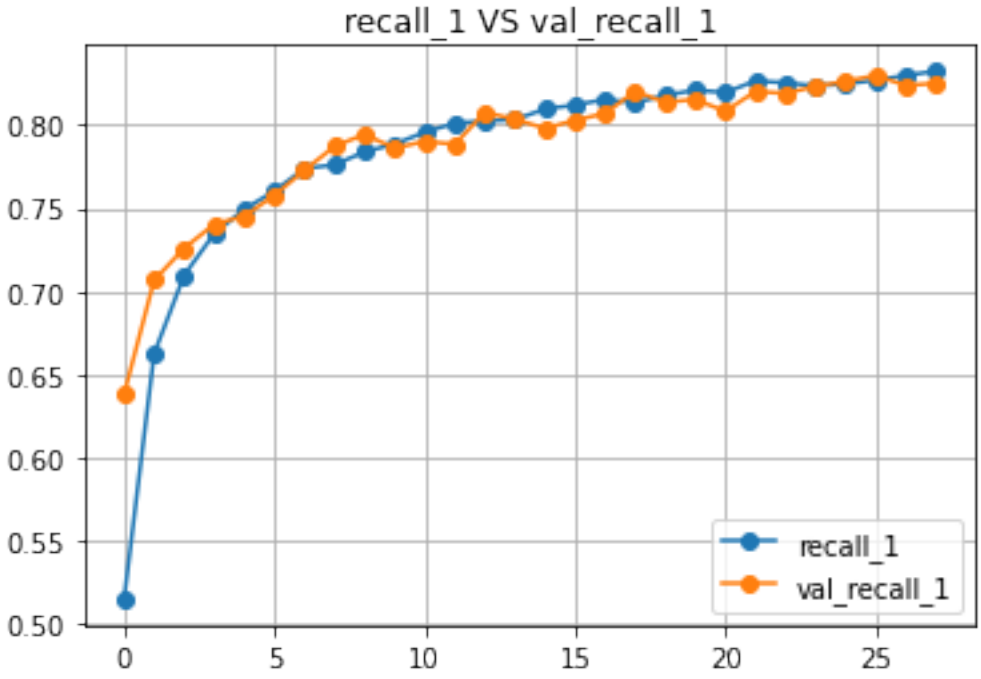
- Creamos una funcion para reutilizar la funcion que genera el grafico, para los modelos que se porbaran mas adelante.

[46]:
```
def plot_metrics(history,metrics=[]): #retorna una lista de tuplas
    fig,axes=plt.subplots(1,len(metrics))
    fig.set_size_inches(15,5)
    graph=pd.DataFrame(history)
    for i,ax in enumerate(axes.flat):
        graph[list(metrics[i])].plot(kind="line",style="o-",ax=ax)
        ax.set_title(" VS ".join(list(metrics[i])))
        ax.grid(True)
    plt.show()
```

[47]:
```
model_history=df_history.to_dict("list")
metrics=[("accuracy","val_accuracy"),("loss","val_loss")]
plot_metrics(model_history,metrics=metrics)
```



[43]:
```
plot_results(model_history,["recall_1","val_recall_1"])
plt.grid(True)
```



- Guardamos el modelo final (3/3) para validarlo.

[48]:
```
model_base.save("baselinev3.h5")
```

- El entrenamiento del modelo se detuvo en 28 epochs lo que nos dice que la funcion de perdida en la data de validacion no mejoro por 3 epochs consecutivos, probablemente, ya no mejore para futuras epocas.

Ahora veamos el rendiemiento del modelo base en los datos de entrenamiento y validacion

[52]:
```
model_base.evaluate(train_generator)
```

```
477/477 [==============================] - 704s 1s/step - loss: 0.3878 -
accuracy: 0.8566 - recall_1: 0.8322
```

[52]:
```
[0.3877873122692108, 0.8566084504127502, 0.8321958184242249]
```

[53]:
```
model_base.evaluate(validation_generator)
```

```
53/53 [==============================] - 81s 2s/step - loss: 0.4038 - accuracy:
0.8512 - recall_1: 0.8253
```

[53]:
```
[0.4038473963737488, 0.8512396812438965, 0.825265645980835]
```

- Mostramos la matriz de confusion en los datos de entrenamiento y validacion

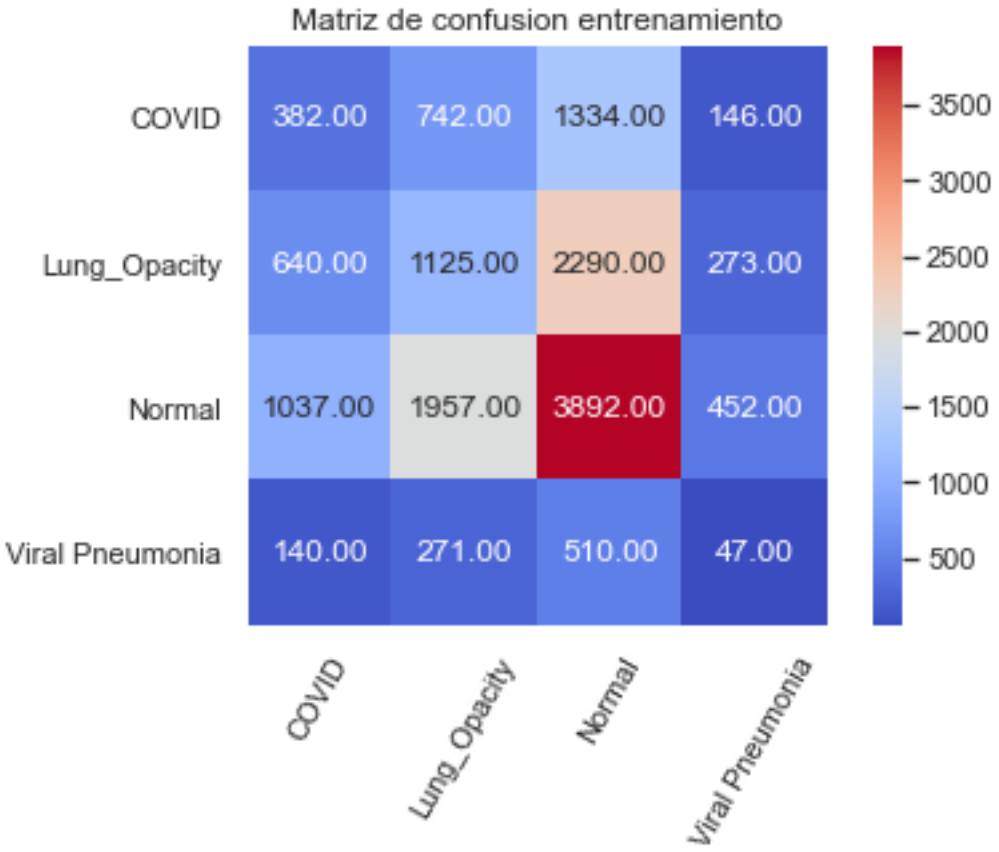
Para el conjunto de entrenamiento

[59]:
```
import numpy as np,seaborn as sns; sns.set()

y_true=train_generator.classes
y_pred=np.argmax(model_base.predict(train_generator),axis=1)
```

[66]:
```
train_generator.class_indices
```

[66]:
```
{'COVID': 0, 'Lung_Opacity': 1, 'Normal': 2, 'Viral Pneumonia': 3}
```

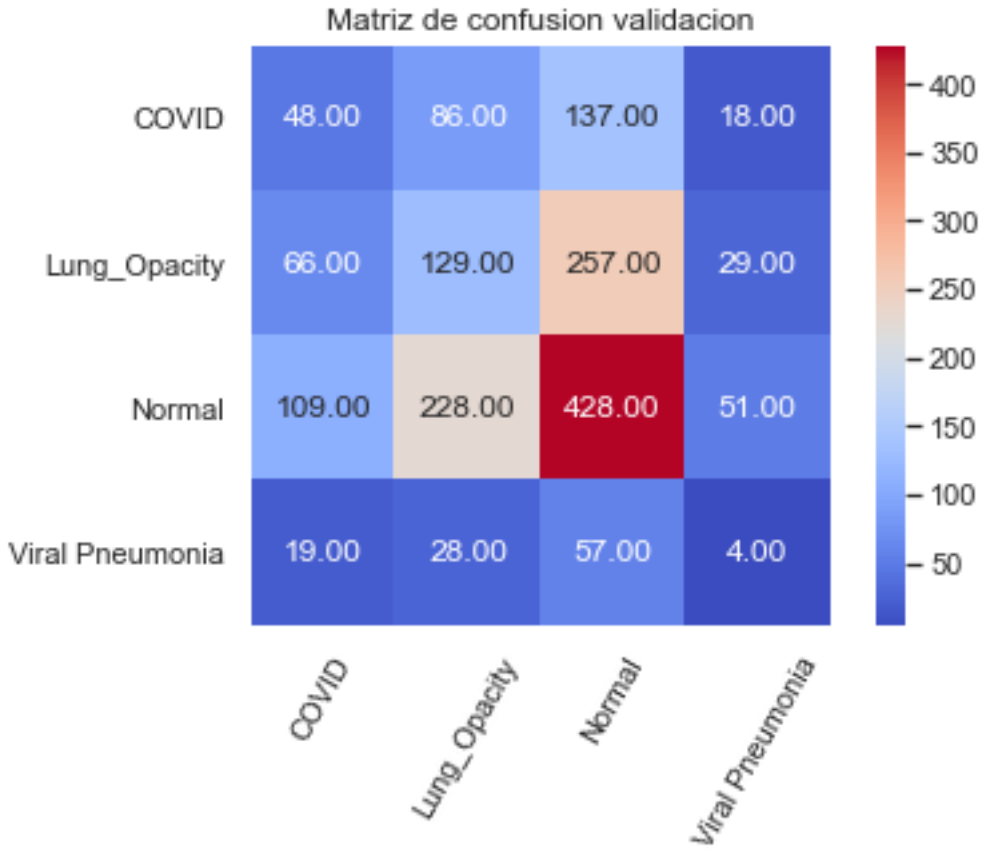
[76]:
```
#obtenemos la matriz de confusion de sklearn
from sklearn.metrics import confusion_matrix

classes=train_generator.class_indices.keys()
mat_train=confusion_matrix(y_true,y_pred)
sns.heatmap(mat_train,square=True,annot=True,fmt="0.
 ↪2f",cmap="coolwarm",xticklabels=classes,yticklabels=classes)
plt.xticks(rotation=60)
plt.title("Matriz de confusion entrenamiento")
plt.show()
```



- Para el conjunto de validacion

[77]:
```
y_true=validation_generator.classes
y_pred=np.argmax(model_base.predict(validation_generator),axis=1)
```

[78]:
```
classes=validation_generator.class_indices.keys()
mat_val=confusion_matrix(y_true,y_pred)
sns.heatmap(mat_val,square=True,annot=True,fmt="0.
 ↪2f",cmap="coolwarm",xticklabels=classes,yticklabels=classes)
plt.xticks(rotation=60)
plt.title("Matriz de confusion validacion")
plt.show()
```



0.0.5 RESULTADOS FINALES: MODELO BASE

- El modelo base ha obtenido un puntaje de accuracy **ACC=85.66%** y recall **RECALL=83.22%** en el **conjunto de entrenamiento**.
- El modelo ha obtendio un puntaje de accuracy **ACC=85.12%** y recall **RECALL=82.52%**

en el **conjunto de validacion**.

IMPORTANTE: El modelo ha alcanzado el equilibrio entre los datos de entrenamiento y validacion, lo que siginifica que es un modelo final con aproximadamente 85% de precision. Este modelo pasará a la **Fase de evaluación del modelo**.

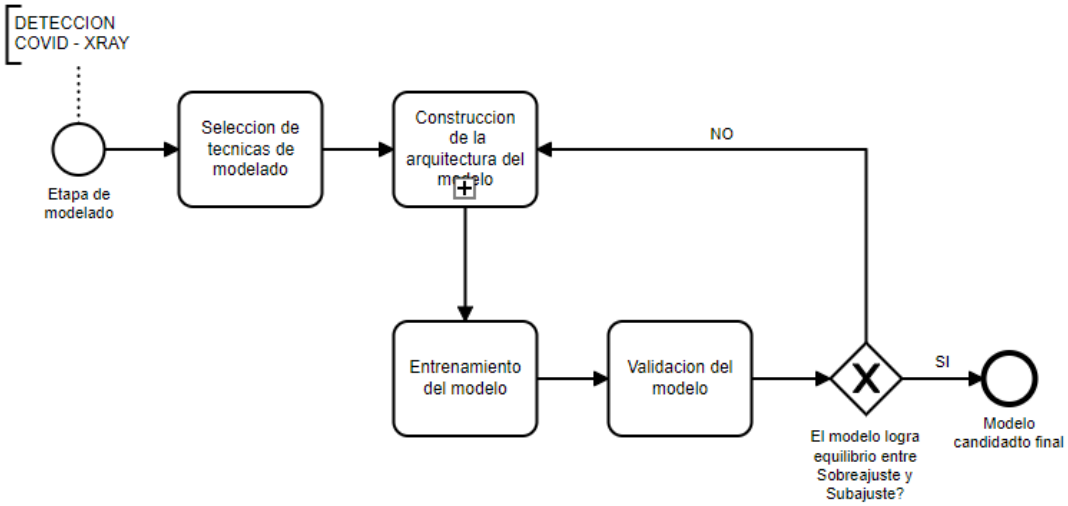
[ ]:

Modelado usando Transfer Learning con DenseNet169 y balanceo por Penalización de clases

August 29, 2021

Esta fase de la metodologia consiste en extraer el valor de los datos desarrollando un modelo que aprenda de los patrones en estos.

- El diagrama en cuestion de esta fase esta a continuacion:



[1]:
```
import tensorflow as tf
from tensorflow.keras.utils import plot_model
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator #generator␣
↪de imagenes
from sklearn.utils import class_weight
from sklearn.metrics import classification_report
import random
```

[2]:
```
tf.keras.backend.clear_session()
```

[3]:
```
tf.random.set_seed(42)#semilla 42, para la reproducibilidad de resultados
random.seed(42)
np.random.seed(42)
```

0.0.1 Canalizacion de datos

- Preparamos la canalizacion de datos, a partir de las imagenes del disco.

[4]:
```
train_datagen=ImageDataGenerator(
            rescale=1.0/255, #escalamos los datos en rangos de [-1,1] El modelo␣
 ↪MobileNetv2 espera esta configuracion
            rotation_range = 45,
            zoom_range = 0.2,
            shear_range = 0.2,
            width_shift_range = 0.2,
            height_shift_range = 0.2,
            horizontal_flip=True,
            vertical_flip = True,
            fill_mode = 'nearest'
        )
#sobre los datos de validacion y test no se hace ningun aumento de datos.
validation_datagen=ImageDataGenerator(rescale=1.0/255) #escalamiento de␣
 ↪validacion a un rango de [0,1]
test_datagen=ImageDataGenerator(rescale=1.0/255)       #escalamiento de test a␣
 ↪un rango de [0,1]
```

[5]:
```
#definimos las rutas para el acceso a los datos
train_path="../input/datasetv3/Datasets/train"
validation_path="../input/datasetv3/Datasets/val"
test_path="../input/datasetv3/Datasets/test"

#creamos los generadores de datos a partir de los flujos de informacion
BATCH_SIZE=32 #tamaño del lote que se ira pasando poco a poco
IMAGE_SIZE=(256,256)

train_generator=train_datagen.flow_from_directory(
    train_path,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)

validation_generator=validation_datagen.flow_from_directory(
    validation_path,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
```

```
)

test_generator=test_datagen.flow_from_directory(
    test_path,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)
```

```
Found 15238 images belonging to 3 classes.
Found 1694 images belonging to 3 classes.
Found 4233 images belonging to 4 classes.
```

0.0.2 Tecnica para el tratamiento de datos Desbalanceados. Penalizacion por pesos de clases

penalizando los pesos de las clases mayoritarias a favor de las clases minoritarias.

[6]:
```
from sklearn.utils import class_weight

classes=train_generator.classes
class_weights=class_weight.compute_class_weight("balanced",np.
 ↪unique(classes),classes)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:70:
FutureWarning: Pass classes=[0 1 2], y=[0 0 0 … 2 2 2] as keyword args. From
version 0.25 passing these as positional arguments will result in an error
  FutureWarning)
```

[7]:
```
class_weights=dict(enumerate(class_weights))
print("Los pesos son {0}".format(class_weights))
```

```
Los pesos son {0: 1.950588837685612, 1: 0.9590886203423967, 2:
0.6921958753520487}
```

0.0.3 Selección de tecnicas de Modelado

Al tratarse de un problema de clasificacion de imagenes entre los posibles candidatos tenemos:

- MultiLayer Pereptron: Red neuronal de capas densamente conectadas
- Convolutional Neuronal Network: Red neuronal convolucional.
- Modelos de machine learning clasico (Maquinas de soporte vectorial, arboles de decision e impulso, etc.)

Escogi la red neuronal convolucional porque **aprende de patrones locales** como rasgos pequeños y en bloques de informacion, mientras que el **MLP** aprende de patrones especficos, e decir de todo el espacio de entrada en general.

0.0.4 Construcción de la arquitectura del modelo

- Para la construccion de la arquitectura crearemos un modelo con transferencia de aprendizaje con una arquitectura de red neuronal solida.
- Generando **Callbacks** para detener el entrenamiento cuando no se tienen buenos resultados

[8]:
```
#cuando la funcion de perdida ya no mejora en los datos de validacion.
early_stopping=tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=10,
                                                mode="min",
                                                restore_best_weights=True)
model_checkpoint=tf.keras.callbacks.ModelCheckpoint("base_model_best_check.
 ↪h5",save_best_only=True)
n_epochs=100
```

1 Aplicación de Transfer Learning usando DenseNet169 y tecnica de anti-desbalanceo Penalización de clases

- Se escogio la arquitectura DenseNet169 por tener un mayor precision sobre el conjunto de datos Image.net en la que fue entrenado.

[9]:
```
INPUT_SHAPE=(256,256,3)
BASE_LEARNING_RATE=0.0001
def build_model_transferLearning():
    dense_net169=tf.keras.applications.DenseNet169(
                weights="imagenet",
                input_shape=INPUT_SHAPE,
                include_top=False
            )
    #descongelamos algunas capas
    dense_net169.trainable=False
    for layer in dense_net169.layers:
        if 'conv5' in layer.name:
            layer.trainable=True
        else:
            layer.trainable=False
    #creamos el modelo
    inputs=tf.keras.Input(shape=INPUT_SHAPE)
    x=dense_net169(inputs)
    x=tf.keras.layers.Flatten()(x)

    x=tf.keras.layers.BatchNormalization()(x)
    x=tf.keras.layers.Dense(256,activation="relu")(x)
    x=tf.keras.layers.Dropout(0.4)(x)

    x=tf.keras.layers.BatchNormalization()(x)
    x=tf.keras.layers.Dense(128,activation="relu")(x)
    x=tf.keras.layers.Dropout(0.4)(x)
```

```
    outputs=tf.keras.layers.Dense(3,activation="softmax")(x)

    model=tf.keras.Model(inputs,outputs)

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=BASE_LEARNING_RATE),
        loss="categorical_crossentropy",
        metrics=[
            tf.keras.metrics.CategoricalAccuracy(name="accuracy"),
            tf.keras.metrics.Recall(name="recall")
        ]
    )
    return model
```

[10]:
```
transfer_model=build_model_transferLearning()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/densenet/densenet169_weights_tf_dim_ordering_tf_kernels_notop.h5
51879936/51877672 [==============================] - 0s 0us/step
```

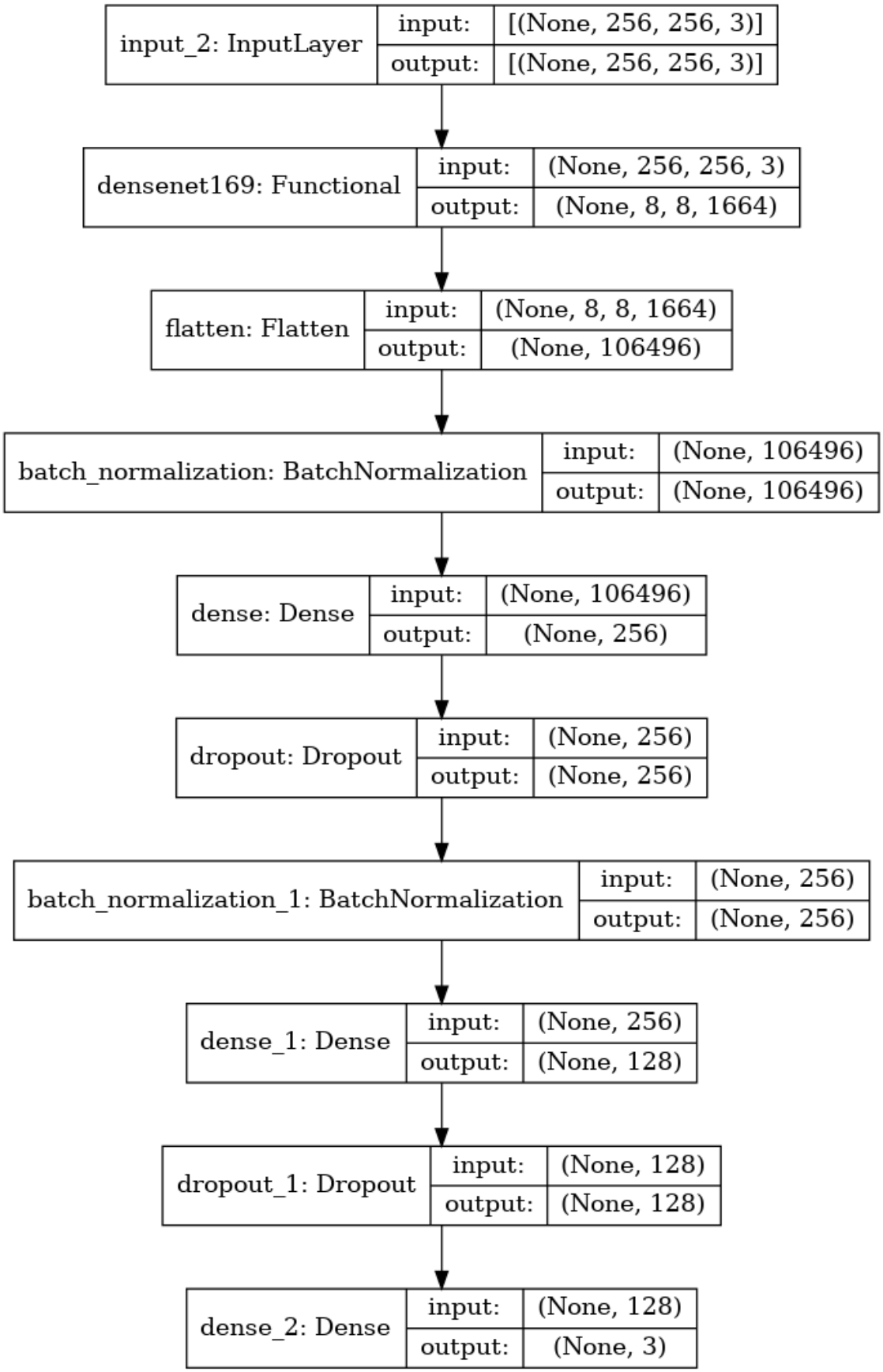
[11]:
```
transfer_model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 256, 256, 3)]     0
_________________________________________________________________
densenet169 (Functional)     (None, 8, 8, 1664)        12642880
_________________________________________________________________
flatten (Flatten)            (None, 106496)            0
_________________________________________________________________
batch_normalization (BatchNo (None, 106496)            425984
_________________________________________________________________
dense (Dense)                (None, 256)               27263232
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 387
=================================================================
Total params: 40,366,403
```

```
Trainable params: 33,423,619
Non-trainable params: 6,942,784
_________________________________________________________________
```

[12]:
```
plot_model(transfer_model,"transfer_densenet.png",show_shapes=True)
```

[12]:


input_2: InputLayer
input: [(None, 256, 256, 3)]
output: [(None, 256, 256, 3)]
densenet169: Functional
input: (None, 256, 256, 3)
output: (None, 8, 8, 1664)
flatten: Flatten
input: (None, 8, 8, 1664)
output: (None, 106496)
batch_normalization: BatchNormalization
input: (None, 106496)
output: (None, 106496)
dense: Dense
input: (None, 106496)
output: (None, 256)
dropout: Dropout
input: (None, 256)
output: (None, 256)
batch_normalization_1: BatchNormalization
input: (None, 256)
output: (None, 256)
dense_1: Dense
input: (None, 256)
output: (None, 128)
dropout_1: Dropout
input: (None, 128)
output: (None, 128)
dense_2: Dense
input: (None, 128)
output: (None, 3)

ENTRENAMIENTO DEL MODELO

- El entrenamiento se realiza en **100 epocas**, un generador de datos de entrenamiento, un generador de datos de validacion, 2 callbacks para detener el entrenamiento de manera temprana en caso no se obtengan buenos resultados en base a la funcion de perdida en los datos de validacion durante 10 epocas consecutivas, y otra para guardar por puntos el mejor modelo obtenido hasta el momento.

[13]:
```
transfer_model=build_model_transferLearning()
history_model=transfer_model.fit(
                    train_generator,
                    epochs=n_epochs,
                    validation_data=validation_generator,
                    validation_steps=validation_generator.samples//
 ↪validation_generator.batch_size,
                    callbacks=[early_stopping,model_checkpoint],
                    class_weight=class_weights,#penalizacion de pesos para el␣
 ↪balanceo,
                    workers=8
            )
```

```
Epoch 1/100
477/477 [==============================] - 262s 503ms/step - loss: 0.8474 -
accuracy: 0.6627 - recall: 0.6296 - val_loss: 0.5108 - val_accuracy: 0.7969 -
val_recall: 0.7825
Epoch 2/100
477/477 [==============================] - 243s 502ms/step - loss: 0.5319 -
accuracy: 0.7942 - recall: 0.7743 - val_loss: 0.3638 - val_accuracy: 0.8588 -
val_recall: 0.8504
Epoch 3/100
477/477 [==============================] - 238s 494ms/step - loss: 0.4465 -
accuracy: 0.8305 - recall: 0.8121 - val_loss: 0.3318 - val_accuracy: 0.8870 -
val_recall: 0.8678
Epoch 4/100
477/477 [==============================] - 237s 488ms/step - loss: 0.3742 -
accuracy: 0.8566 - recall: 0.8420 - val_loss: 0.3582 - val_accuracy: 0.8576 -
val_recall: 0.8438
Epoch 5/100
477/477 [==============================] - 238s 491ms/step - loss: 0.3737 -
accuracy: 0.8563 - recall: 0.8440 - val_loss: 0.3558 - val_accuracy: 0.8588 -
val_recall: 0.8486
Epoch 6/100
477/477 [==============================] - 237s 491ms/step - loss: 0.3341 -
accuracy: 0.8711 - recall: 0.8596 - val_loss: 0.2772 - val_accuracy: 0.9002 -
val_recall: 0.8912
```

```
Epoch 7/100
477/477 [==============================] - 237s 491ms/step - loss: 0.3295 -
accuracy: 0.8712 - recall: 0.8580 - val_loss: 0.2741 - val_accuracy: 0.9038 -
val_recall: 0.8930
Epoch 8/100
477/477 [==============================] - 233s 482ms/step - loss: 0.3067 -
accuracy: 0.8792 - recall: 0.8682 - val_loss: 0.2921 - val_accuracy: 0.8930 -
val_recall: 0.8840
Epoch 9/100
477/477 [==============================] - 234s 484ms/step - loss: 0.2877 -
accuracy: 0.8868 - recall: 0.8761 - val_loss: 0.2601 - val_accuracy: 0.9062 -
val_recall: 0.9026
Epoch 10/100
477/477 [==============================] - 233s 482ms/step - loss: 0.2809 -
accuracy: 0.8905 - recall: 0.8827 - val_loss: 0.2395 - val_accuracy: 0.9093 -
val_recall: 0.9014
Epoch 11/100
477/477 [==============================] - 231s 478ms/step - loss: 0.2734 -
accuracy: 0.8948 - recall: 0.8848 - val_loss: 0.2476 - val_accuracy: 0.9069 -
val_recall: 0.9008
Epoch 12/100
477/477 [==============================] - 234s 484ms/step - loss: 0.2701 -
accuracy: 0.8989 - recall: 0.8909 - val_loss: 0.2741 - val_accuracy: 0.8978 -
val_recall: 0.8936
Epoch 13/100
477/477 [==============================] - 235s 484ms/step - loss: 0.2690 -
accuracy: 0.8937 - recall: 0.8862 - val_loss: 0.2466 - val_accuracy: 0.9147 -
val_recall: 0.9129
Epoch 14/100
477/477 [==============================] - 234s 484ms/step - loss: 0.2608 -
accuracy: 0.8998 - recall: 0.8940 - val_loss: 0.2942 - val_accuracy: 0.8918 -
val_recall: 0.8888
Epoch 15/100
477/477 [==============================] - 234s 483ms/step - loss: 0.2439 -
accuracy: 0.9051 - recall: 0.8991 - val_loss: 0.2441 - val_accuracy: 0.9087 -
val_recall: 0.9056
Epoch 16/100
477/477 [==============================] - 235s 487ms/step - loss: 0.2416 -
accuracy: 0.9033 - recall: 0.8981 - val_loss: 0.2478 - val_accuracy: 0.9099 -
val_recall: 0.9069
Epoch 17/100
477/477 [==============================] - 234s 485ms/step - loss: 0.2251 -
accuracy: 0.9124 - recall: 0.9052 - val_loss: 0.2949 - val_accuracy: 0.8948 -
val_recall: 0.8900
Epoch 18/100
477/477 [==============================] - 238s 493ms/step - loss: 0.2179 -
accuracy: 0.9183 - recall: 0.9131 - val_loss: 0.2675 - val_accuracy: 0.8996 -
val_recall: 0.8936
```

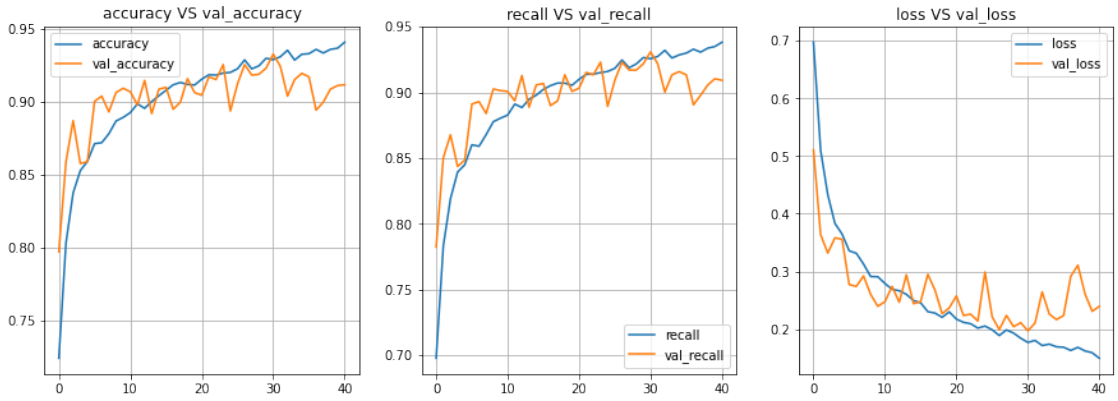
```
Epoch 19/100
477/477 [==============================] - 237s 491ms/step - loss: 0.2296 -
accuracy: 0.9099 - recall: 0.9054 - val_loss: 0.2267 - val_accuracy: 0.9159 -
val_recall: 0.9135
Epoch 20/100
477/477 [==============================] - 237s 490ms/step - loss: 0.2266 -
accuracy: 0.9135 - recall: 0.9073 - val_loss: 0.2368 - val_accuracy: 0.9062 -
val_recall: 0.9008
Epoch 21/100
477/477 [==============================] - 235s 484ms/step - loss: 0.2215 -
accuracy: 0.9156 - recall: 0.9105 - val_loss: 0.2575 - val_accuracy: 0.9044 -
val_recall: 0.9032
Epoch 22/100
477/477 [==============================] - 235s 487ms/step - loss: 0.2032 -
accuracy: 0.9225 - recall: 0.9186 - val_loss: 0.2236 - val_accuracy: 0.9171 -
val_recall: 0.9153
Epoch 23/100
477/477 [==============================] - 235s 487ms/step - loss: 0.2054 -
accuracy: 0.9212 - recall: 0.9167 - val_loss: 0.2259 - val_accuracy: 0.9153 -
val_recall: 0.9135
Epoch 24/100
477/477 [==============================] - 236s 487ms/step - loss: 0.1993 -
accuracy: 0.9220 - recall: 0.9174 - val_loss: 0.2139 - val_accuracy: 0.9255 -
val_recall: 0.9231
Epoch 25/100
477/477 [==============================] - 237s 490ms/step - loss: 0.2040 -
accuracy: 0.9224 - recall: 0.9182 - val_loss: 0.2990 - val_accuracy: 0.8936 -
val_recall: 0.8894
Epoch 26/100
477/477 [==============================] - 236s 488ms/step - loss: 0.1930 -
accuracy: 0.9250 - recall: 0.9208 - val_loss: 0.2213 - val_accuracy: 0.9123 -
val_recall: 0.9093
Epoch 27/100
477/477 [==============================] - 235s 486ms/step - loss: 0.1893 -
accuracy: 0.9302 - recall: 0.9259 - val_loss: 0.1991 - val_accuracy: 0.9255 -
val_recall: 0.9231
Epoch 28/100
477/477 [==============================] - 236s 488ms/step - loss: 0.1894 -
accuracy: 0.9247 - recall: 0.9204 - val_loss: 0.2237 - val_accuracy: 0.9183 -
val_recall: 0.9171
Epoch 29/100
477/477 [==============================] - 235s 487ms/step - loss: 0.1940 -
accuracy: 0.9236 - recall: 0.9205 - val_loss: 0.2041 - val_accuracy: 0.9189 -
val_recall: 0.9171
Epoch 30/100
477/477 [==============================] - 235s 487ms/step - loss: 0.1801 -
accuracy: 0.9316 - recall: 0.9283 - val_loss: 0.2111 - val_accuracy: 0.9231 -
val_recall: 0.9219
```

```
Epoch 31/100
477/477 [==============================] - 235s 486ms/step - loss: 0.1753 -
accuracy: 0.9264 - recall: 0.9236 - val_loss: 0.1969 - val_accuracy: 0.9327 -
val_recall: 0.9309
Epoch 32/100
477/477 [==============================] - 238s 494ms/step - loss: 0.1730 -
accuracy: 0.9321 - recall: 0.9294 - val_loss: 0.2105 - val_accuracy: 0.9243 -
val_recall: 0.9219
Epoch 33/100
477/477 [==============================] - 239s 494ms/step - loss: 0.1785 -
accuracy: 0.9370 - recall: 0.9337 - val_loss: 0.2645 - val_accuracy: 0.9038 -
val_recall: 0.9002
Epoch 34/100
477/477 [==============================] - 240s 496ms/step - loss: 0.1732 -
accuracy: 0.9300 - recall: 0.9274 - val_loss: 0.2259 - val_accuracy: 0.9153 -
val_recall: 0.9135
Epoch 35/100
477/477 [==============================] - 238s 492ms/step - loss: 0.1689 -
accuracy: 0.9348 - recall: 0.9310 - val_loss: 0.2165 - val_accuracy: 0.9195 -
val_recall: 0.9159
Epoch 36/100
477/477 [==============================] - 240s 496ms/step - loss: 0.1632 -
accuracy: 0.9361 - recall: 0.9322 - val_loss: 0.2236 - val_accuracy: 0.9171 -
val_recall: 0.9135
Epoch 37/100
477/477 [==============================] - 239s 495ms/step - loss: 0.1687 -
accuracy: 0.9353 - recall: 0.9318 - val_loss: 0.2914 - val_accuracy: 0.8942 -
val_recall: 0.8906
Epoch 38/100
477/477 [==============================] - 238s 492ms/step - loss: 0.1723 -
accuracy: 0.9324 - recall: 0.9297 - val_loss: 0.3107 - val_accuracy: 0.8996 -
val_recall: 0.8978
Epoch 39/100
477/477 [==============================] - 238s 494ms/step - loss: 0.1552 -
accuracy: 0.9381 - recall: 0.9361 - val_loss: 0.2609 - val_accuracy: 0.9087 -
val_recall: 0.9056
Epoch 40/100
477/477 [==============================] - 237s 490ms/step - loss: 0.1611 -
accuracy: 0.9352 - recall: 0.9333 - val_loss: 0.2308 - val_accuracy: 0.9111 -
val_recall: 0.9105
Epoch 41/100
477/477 [==============================] - 239s 495ms/step - loss: 0.1488 -
accuracy: 0.9416 - recall: 0.9391 - val_loss: 0.2395 - val_accuracy: 0.9117 -
val_recall: 0.9093
```

[14]:
```
transfer_model.save("./
↪tranferlearning_densenet169_with_balanced_focal_loss_3_class.h5")
```

[15]:
```
def plot_metrics(history,metrics=[]): #retorna una lista de tuplas
    fig,axes=plt.subplots(1,len(metrics))
    fig.set_size_inches(15,5)
    graph=pd.DataFrame(history)
    for i,ax in enumerate(axes.flat):
        graph[list(metrics[i])].plot(kind="line",style="-",ax=ax)
        ax.set_title(" VS ".join(list(metrics[i])))
        ax.grid(True)
    plt.show()
```

[16]:
```
metrics=[("accuracy","val_accuracy"),("recall","val_recall"),("loss","val_loss")]
plot_metrics(history_model.history,metrics=metrics)
```



[17]:
```
#obtenemos el numero de epocas donde se detuvo y lo configuramos como un epoch␣
 ↪inicial para el siguiente
#entrenamiento
EPOCH_STOP=len(history_model.epoch)
print("El modelo se entreno en",EPOCH_STOP,"Epochs")
```

```
El modelo se entreno en 41 Epochs
```

- El entrenamiento del modelo se detuvo en 47 epochs lo que nos dice que la funcion de perdida en la data de validacion no mejoro por 10 epochs consecutivos, probablemente, ya no mejore para futuras epocas.

Ahora veamos el rendimiento del modelo base en los datos de entrenamiento y validacion

[18]:
```
transfer_model.evaluate(train_generator)
```

```
477/477 [==============================] - 253s 530ms/step - loss: 0.1678 -
accuracy: 0.9421 - recall: 0.9387
```

[18]:
```
[0.16779577732086182, 0.9420527815818787, 0.9387058615684509]
```

[19]:
```
transfer_model.evaluate(validation_generator)
```

```
53/53 [==============================] - 8s 149ms/step - loss: 0.1943 -
accuracy: 0.9339 - recall: 0.9321
```

[19]:
```
[0.19431829452514648, 0.93388432264328, 0.9321133494377136]
```

- Mostramos la matriz de confusion en los datos de entrenamiento y validacion

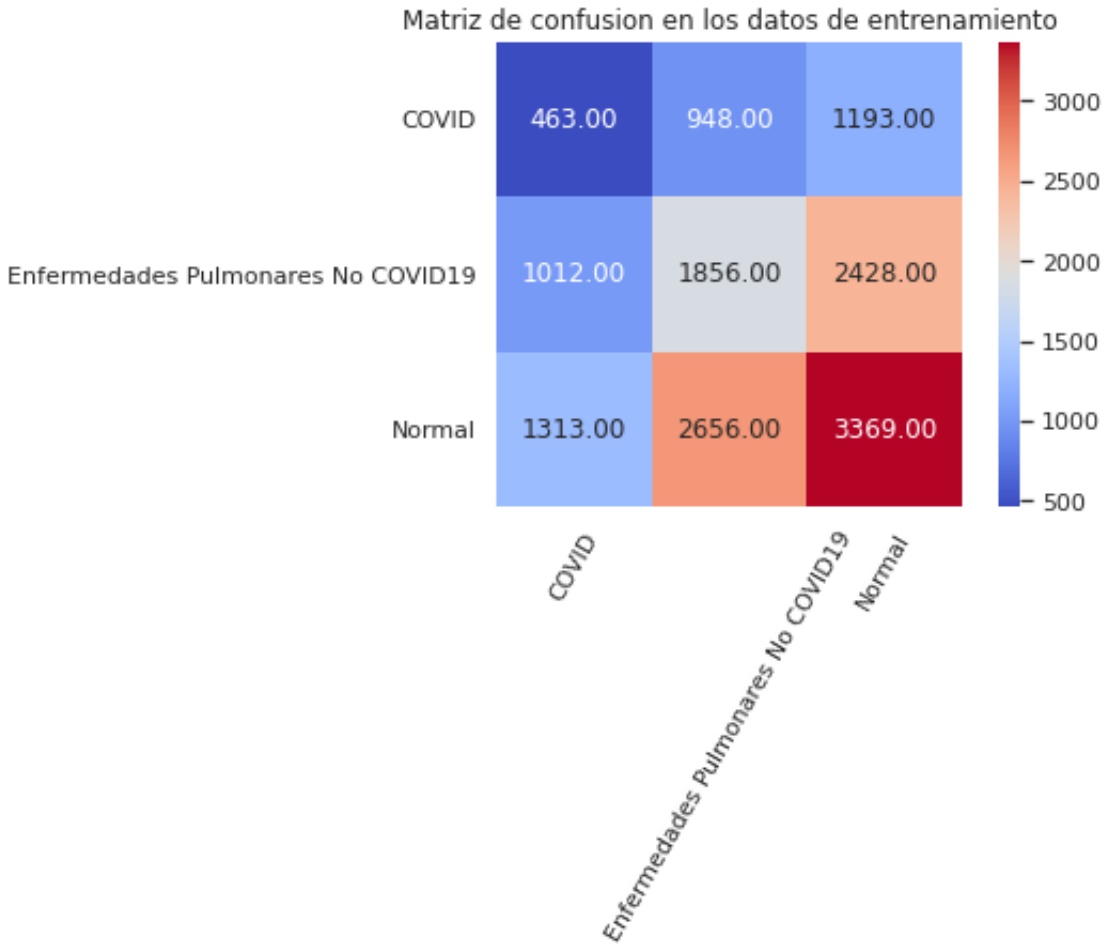
Para el conjunto de entrenamiento

[20]:
```
y_true=train_generator.classes
predictions=transfer_model.predict(train_generator)
y_pred=np.argmax(predictions,axis=1)
```

[21]:
```
print("Indices de clase")
for idx,clase in train_generator.class_indices.items():
    print(idx,":",clase)
```

```
Indices de clase
COVID : 0
Enfermedades Pulmonares No COVID19 : 1
Normal : 2
```

[22]:
```
#obtenemos la matriz de confusion de sklearn
from sklearn.metrics import confusion_matrix
import seaborn as sns; sns.set()
classes=train_generator.class_indices.keys()
mat_train=confusion_matrix(y_true,y_pred)
sns.heatmap(mat_train,square=True,annot=True,fmt="0.
 ↪2f",cmap="coolwarm",xticklabels=classes,yticklabels=classes)
plt.xticks(rotation=60)
plt.title("Matriz de confusion en los datos de entrenamiento")
plt.show()
```



- Reporte de clasificacion para el conjunto de entrenamiento

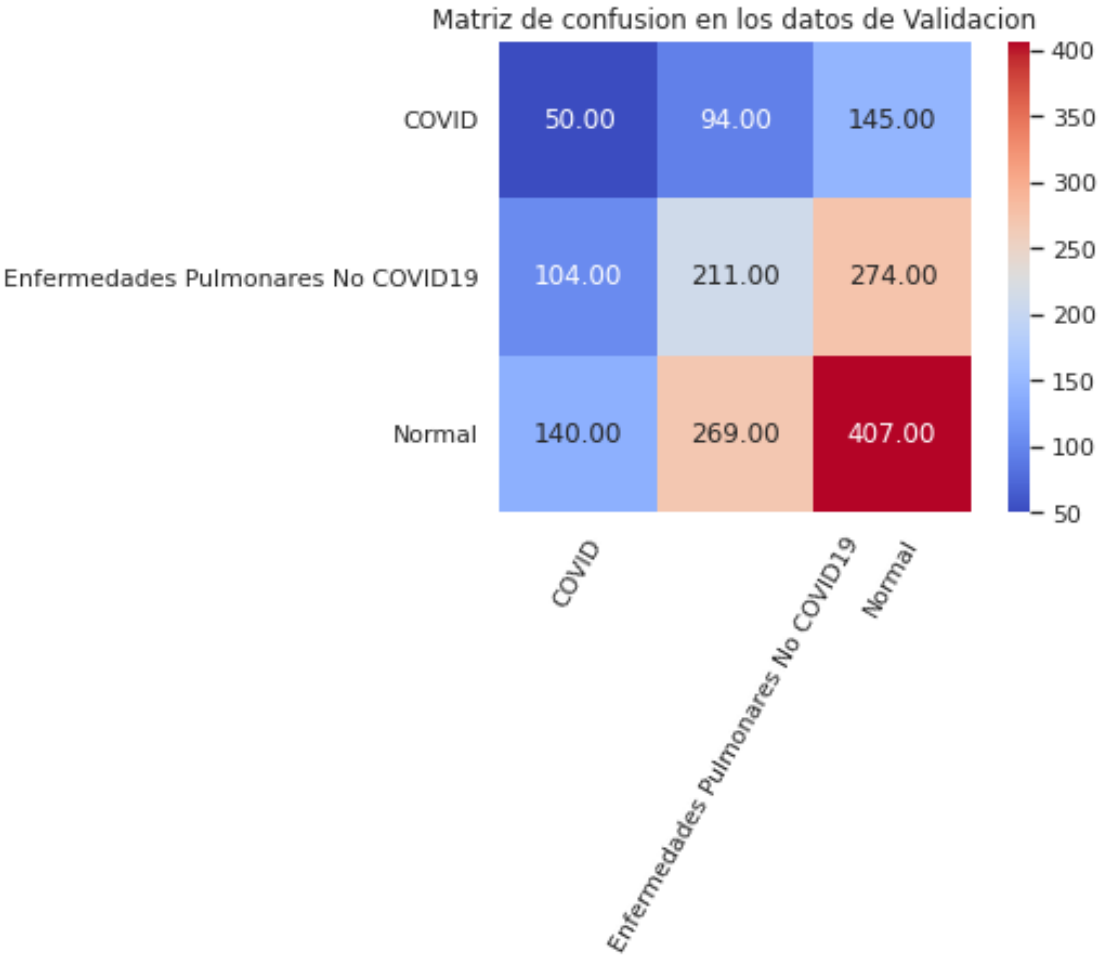
[23]:
```
report=classification_report(y_true,y_pred,target_names=list(train_generator.
↪class_indices.keys()))
print(report)
```

```
                                precision    recall  f1-score   support

                        COVID       0.17      0.18      0.17      2604
Enfermedades Pulmonares No COVID19       0.34      0.35      0.35      5296
                       Normal       0.48      0.46      0.47      7338

                     accuracy                           0.37     15238
                    macro avg       0.33      0.33      0.33     15238
                 weighted avg       0.38      0.37      0.38     15238
```

- Para el conjunto de validacion

[24]:
```
y_true=validation_generator.classes
predictions=transfer_model.predict(validation_generator)
y_pred=np.argmax(predictions,axis=1)
```

[25]:
```
classes=validation_generator.class_indices.keys()
mat_val=confusion_matrix(y_true,y_pred)
sns.heatmap(mat_val,square=True,annot=True,fmt="0.
 ↪2f",cmap="coolwarm",xticklabels=classes,yticklabels=classes)
plt.xticks(rotation=60)
plt.title("Matriz de confusion en los datos de Validacion")
plt.show()
```



- Reporte de clasificacion para los datos de validacion

[26]:
```
report=classification_report(y_true,y_pred,target_names=list(train_generator.
 ↪class_indices.keys()))
print(report)
```

	precision	recall	f1-score	support
COVID	0.17	0.17	0.17	289
Enfermedades Pulmonares No COVID19	0.37	0.36	0.36	589
Normal	0.49	0.50	0.50	816
accuracy			0.39	1694
macro avg	0.34	0.34	0.34	1694
weighted avg	0.39	0.39	0.39	1694

1.0.1 RESULTADOS FINALES: MODELO TRANFER LEARNING CON PENALIZACION DE PESOS PARA EL BALANCEO DE CLASES. USANDO 3 CLASES

- El modelo base ha obtenido un puntaje de accuracy **ACC=94.21%** y recall **RECALL=93.87%** en el **conjunto de entrenamiento**.
- El modelo ha obtendio un puntaje de accuracy **ACC=93.39%** y recall **RECALL=93.21%** en el **conjunto de validacion**.

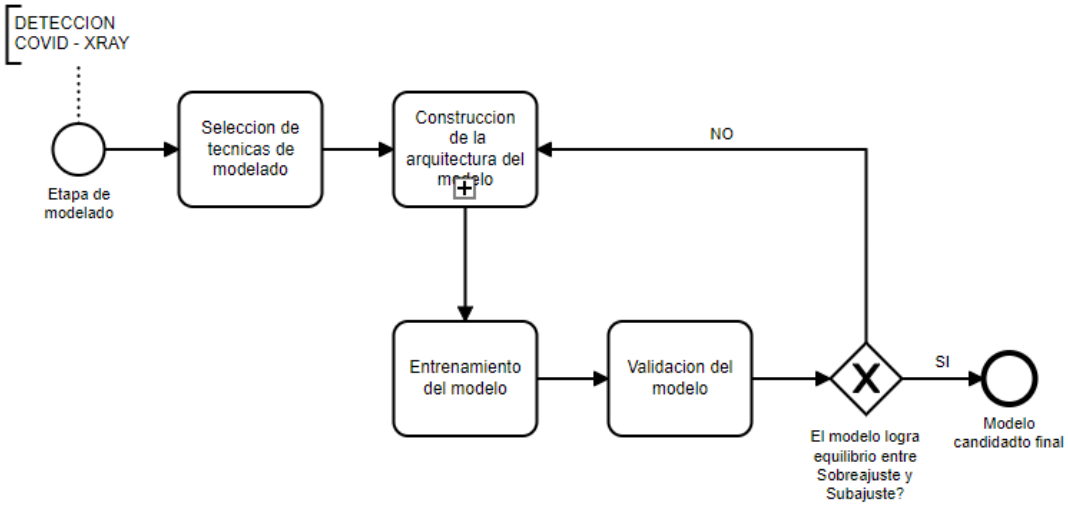
IMPORTANTE: El modelo ha alcanzado equilibrio entre los datos de entrenamiento y validacion, lo que significa que es un modelo final con aproximadamente 93% de precision. Este modelo pasará a la **Fase de evaluación del modelo**.

Modelado usando Transfer Learning con DenseNet169 y balanceo por Perdida Focal

August 29, 2021

Esta fase de la metdologia consiste en extraer el valor de los datos desarrollando un modelo que aprenda de los patrones en estos.

- El diagrama en cuestion de esta fase esta a continuacion:



[1]:
```
import tensorflow as tf
from tensorflow.keras.utils import plot_model
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator #generator␣
↪de imagenes
from sklearn.utils import class_weight
from sklearn.metrics import classification_report
import random
```

[2]:
```
tf.keras.backend.clear_session()
```

[3]:
```
tf.random.set_seed(42)#semilla 42, para la reproducibilidad de resultados
random.seed(42)
np.random.seed(42)
```

0.0.1 Canalizacion de datos

- Preparamos la canalizacion de datos, a partir de las imagenes del disco.

[4]:
```
train_datagen=ImageDataGenerator(
            rescale=1.0/255, #escalamos los datos en rangos de [-1,1] El modelo␣
 ↪MobileNetv2 espera esta configuracion
            rotation_range = 45,
            zoom_range = 0.2,
            shear_range = 0.2,
            width_shift_range = 0.2,
            height_shift_range = 0.2,
            horizontal_flip=True,
            vertical_flip = True,
            fill_mode = 'nearest'
        )
#sobre los datos de validacion y test no se hace ningun aumento de datos.
validation_datagen=ImageDataGenerator(rescale=1.0/255) #escalamiento de␣
 ↪validacion a un rango de [0,1]
test_datagen=ImageDataGenerator(rescale=1.0/255)       #escalamiento de test a␣
 ↪un rango de [0,1]
```

[5]:
```
#definimos las rutas para el acceso a los datos
train_path="../input/datasetv3/Datasets/train"
validation_path="../input/datasetv3/Datasets/val"
test_path="../input/datasetv3/Datasets/test"

#creamos los generadores de datos a partir de los flujos de informacion
BATCH_SIZE=32 #tamaño del lote que se ira pasando poco a poco
IMAGE_SIZE=(256,256)

train_generator=train_datagen.flow_from_directory(
    train_path,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)

validation_generator=validation_datagen.flow_from_directory(
    validation_path,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
```

```
)

test_generator=test_datagen.flow_from_directory(
    test_path,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)
```

```
Found 15238 images belonging to 3 classes.
Found 1694 images belonging to 3 classes.
Found 4233 images belonging to 4 classes.
```

0.0.2 Tecnica para el tratamiento de datos Desbalaceados. Focal Loss Categorical

- Para ello personalizaremos una funcion de costo
- En nuestro caso trabajamos con imagenes, y tenemos la opcion de tratar este desbalanceo **penalizando los pesos de las clases** mayoritarias a favor de las clases minoritarias.

[6]:
```
def focal_loss(gamma=2., alpha=4.):

    gamma = float(gamma)
    alpha = float(alpha)

    def focal_loss_fixed(y_true, y_pred):
        """Focal loss for multi-classification
        FL(p_t)=-alpha(1-p_t)^{gamma}ln(p_t)
        Notice: y_pred is probability after softmax
        gradient is d(Fl)/d(p_t) not d(Fl)/d(x) as described in paper
        d(Fl)/d(p_t) * [p_t(1-p_t)] = d(Fl)/d(x)
        Focal Loss for Dense Object Detection
        https://arxiv.org/abs/1708.02002
        Arguments:
            y_true {tensor} -- ground truth labels, shape of [batch_size,␣
 ↪num_cls]
            y_pred {tensor} -- model's output, shape of [batch_size, num_cls]
        Keyword Arguments:
            gamma {float} -- (default: {2.0})
            alpha {float} -- (default: {4.0})
        Returns:
            [tensor] -- loss.
        """
        epsilon = 1.e-9
        y_true = tf.convert_to_tensor(y_true, tf.float32)
        y_pred = tf.convert_to_tensor(y_pred, tf.float32)

        model_out = tf.add(y_pred, epsilon)
        ce = tf.multiply(y_true, -tf.math.log(model_out))
```

```
        weight = tf.multiply(y_true, tf.pow(tf.subtract(1., model_out), gamma))
        fl = tf.multiply(alpha, tf.multiply(weight, ce))
        reduced_fl = tf.reduce_max(fl, axis=1)
        return tf.reduce_mean(reduced_fl)
    return focal_loss_fixed
```

0.0.3 Selección de tecnicas de Modelado

Al tratarse de un problema de clasificacion de imagenes entre los posibles candidatos tenemos:

- MultiLayer Pereptron: Red neuronal de capas densamente conectadas
- Convolutional Neuronal Network: Red neuronal convolucional.
- Modelos de machine learning clasico (Maquinas de soporte vectorial, arboles de decision e impulso, etc.)

Escogi la red neuronal convolucional porque **aprende de patrones locales** como rasgos pequeños y en bloques de informacion, mientras que el **MLP** aprende de patrones especficos, e decir de todo el espacio de entrada en general.

0.0.4 Construcción de la arquitectura del modelo

- Para la construccion de la arquitectura crearemos un modelo desde 0 con una arquitectura de red neuronal solida.

Una vez obtenidos la arquitectura del modelo * Generando **Callbacks** para detener el entrenamiento cuando no se tienen buenos resultados

[7]:
```
#cuando la funcion de perdida ya no mejora en los datos de validacion.
early_stopping=tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=10,
                                                mode="min",
                                                restore_best_weights=True)
model_checkpoint=tf.keras.callbacks.ModelCheckpoint("base_model_best_check.
 ↪h5",save_best_only=True)
n_epochs=100
```

1 Aplicación de Transfer Learning usando DenseNet169 y tecnica de anti-desbalanceo Focal Loss Categorical

- Se escogio la arquitectura DenseNet169 por tener un mayor precision sobre el conjunto de datos Image.net en la que fue entrenado.

[8]:
```
INPUT_SHAPE=(256,256,3)
BASE_LEARNING_RATE=0.0001
def build_model_transferLearning():
    dense_net169=tf.keras.applications.DenseNet169(
                weights="imagenet",
                input_shape=INPUT_SHAPE,
                include_top=False
            )
```

```
    #descongelamos algunas capas
    dense_net169.trainable=True
    for layer in dense_net169.layers:
        if 'conv5' in layer.name:
            layer.trainable=True
        else:
            layer.trainable=False
    #creamos el modelo
    inputs=tf.keras.Input(shape=INPUT_SHAPE)
    x=dense_net169(inputs)
    x=tf.keras.layers.Flatten()(x)

    x=tf.keras.layers.BatchNormalization()(x)
    x=tf.keras.layers.Dense(256,activation="relu")(x)
    x=tf.keras.layers.Dropout(0.4)(x)

    x=tf.keras.layers.BatchNormalization()(x)
    x=tf.keras.layers.Dense(128,activation="relu")(x)
    x=tf.keras.layers.Dropout(0.4)(x)

    outputs=tf.keras.layers.Dense(3,activation="softmax")(x)

    model=tf.keras.Model(inputs,outputs)

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=BASE_LEARNING_RATE),
        loss=focal_loss(alpha=1.0),
        metrics=[
            tf.keras.metrics.CategoricalAccuracy(name="accuracy"),
            tf.keras.metrics.Recall(name="recall")
        ]
    )
    return model
```

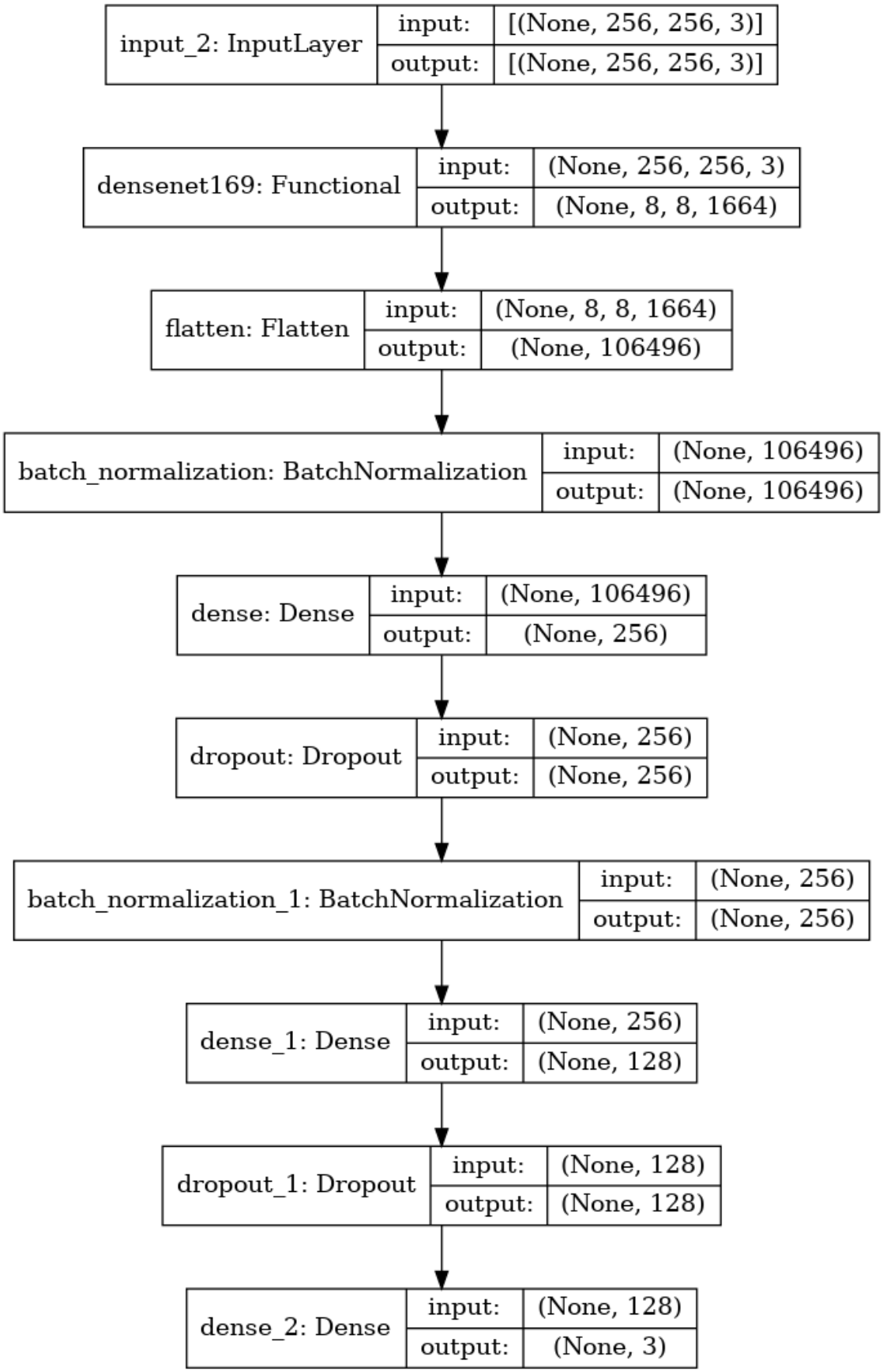
[9]:
```
transfer_model=build_model_transferLearning()
```

[10]:
```
transfer_model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 256, 256, 3)]     0
_________________________________________________________________
densenet169 (Functional)     (None, 8, 8, 1664)        12642880
_________________________________________________________________
flatten (Flatten)            (None, 106496)            0
_________________________________________________________________
```

```
batch_normalization (BatchNo (None, 106496)            425984
_________________________________________________________________
dense (Dense)                (None, 256)               27263232
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 387
=================================================================
Total params: 40,366,403
Trainable params: 33,423,619
Non-trainable params: 6,942,784
_________________________________________________________________
```

[11]:
```
plot_model(transfer_model,"transfer_densenet.png",show_shapes=True)
```
[11]:


input_2: InputLayer
input: [(None, 256, 256, 3)]
output: [(None, 256, 256, 3)]
densenet169: Functional
input: (None, 256, 256, 3)
output: (None, 8, 8, 1664)
flatten: Flatten
input: (None, 8, 8, 1664)
output: (None, 106496)
batch_normalization: BatchNormalization
input: (None, 106496)
output: (None, 106496)
dense: Dense
input: (None, 106496)
output: (None, 256)
dropout: Dropout
input: (None, 256)
output: (None, 256)
batch_normalization_1: BatchNormalization
input: (None, 256)
output: (None, 256)
dense_1: Dense
input: (None, 256)
output: (None, 128)
dropout_1: Dropout
input: (None, 128)
output: (None, 128)
dense_2: Dense
input: (None, 128)
output: (None, 3)

ENTRENAMIENTO DEL MODELO

- El entrenamiento se realiza en **100 epocas**, un generador de datos de entrenamiento, un generador de datos de validacion, 2 callbacks para detener el entrenamiento de manera temprana en caso no se obtengan buenos resultados en base a la funcion de perdida en los datos de validacion durante 10 epocas consecutivas, y otra para guardar por puntos el mejor modelo obtenido hasta el momento.

[12]:
```
transfer_model=build_model_transferLearning()
history_model=transfer_model.fit(
                    train_generator,
                    epochs=n_epochs,
                    validation_data=validation_generator,
                    validation_steps=validation_generator.samples//
 ↪validation_generator.batch_size,
                    callbacks=[early_stopping,model_checkpoint],
                    #class_weight=class_weights,#penalizacion de pesos para el␣
 ↪balanceo,
                    workers=8
            )
```

```
Epoch 1/100
477/477 [==============================] - 252s 492ms/step - loss: 0.5307 -
accuracy: 0.6529 - recall: 0.6065 - val_loss: 0.1964 - val_accuracy: 0.8540 -
val_recall: 0.8305
Epoch 2/100
477/477 [==============================] - 236s 490ms/step - loss: 0.2916 -
accuracy: 0.7745 - recall: 0.7330 - val_loss: 0.1509 - val_accuracy: 0.8438 -
val_recall: 0.7704
Epoch 3/100
477/477 [==============================] - 234s 484ms/step - loss: 0.2221 -
accuracy: 0.8032 - recall: 0.7576 - val_loss: 0.1360 - val_accuracy: 0.8630 -
val_recall: 0.8269
Epoch 4/100
477/477 [==============================] - 230s 477ms/step - loss: 0.1927 -
accuracy: 0.8192 - recall: 0.7781 - val_loss: 0.1048 - val_accuracy: 0.8930 -
val_recall: 0.8564
Epoch 5/100
477/477 [==============================] - 233s 482ms/step - loss: 0.1743 -
accuracy: 0.8372 - recall: 0.7931 - val_loss: 0.1109 - val_accuracy: 0.8756 -
val_recall: 0.8287
Epoch 6/100
477/477 [==============================] - 233s 483ms/step - loss: 0.1487 -
accuracy: 0.8484 - recall: 0.8061 - val_loss: 0.1421 - val_accuracy: 0.8299 -
val_recall: 0.7855
```

```
Epoch 7/100
477/477 [==============================] - 236s 488ms/step - loss: 0.1386 -
accuracy: 0.8592 - recall: 0.8193 - val_loss: 0.0940 - val_accuracy: 0.8978 -
val_recall: 0.8756
Epoch 8/100
477/477 [==============================] - 232s 480ms/step - loss: 0.1263 -
accuracy: 0.8562 - recall: 0.8211 - val_loss: 0.1051 - val_accuracy: 0.8600 -
val_recall: 0.8347
Epoch 9/100
477/477 [==============================] - 231s 479ms/step - loss: 0.1277 -
accuracy: 0.8674 - recall: 0.8354 - val_loss: 0.0973 - val_accuracy: 0.8816 -
val_recall: 0.8534
Epoch 10/100
477/477 [==============================] - 232s 480ms/step - loss: 0.1201 -
accuracy: 0.8688 - recall: 0.8392 - val_loss: 0.1123 - val_accuracy: 0.8804 -
val_recall: 0.8576
Epoch 11/100
477/477 [==============================] - 231s 478ms/step - loss: 0.1144 -
accuracy: 0.8750 - recall: 0.8404 - val_loss: 0.1042 - val_accuracy: 0.8900 -
val_recall: 0.8666
Epoch 12/100
477/477 [==============================] - 232s 480ms/step - loss: 0.1144 -
accuracy: 0.8759 - recall: 0.8491 - val_loss: 0.0943 - val_accuracy: 0.8936 -
val_recall: 0.8546
Epoch 13/100
477/477 [==============================] - 233s 482ms/step - loss: 0.1117 -
accuracy: 0.8834 - recall: 0.8607 - val_loss: 0.0912 - val_accuracy: 0.9069 -
val_recall: 0.8720
Epoch 14/100
477/477 [==============================] - 232s 480ms/step - loss: 0.0990 -
accuracy: 0.8944 - recall: 0.8730 - val_loss: 0.0889 - val_accuracy: 0.9075 -
val_recall: 0.8846
Epoch 15/100
477/477 [==============================] - 230s 477ms/step - loss: 0.0952 -
accuracy: 0.8912 - recall: 0.8695 - val_loss: 0.0923 - val_accuracy: 0.8978 -
val_recall: 0.8510
Epoch 16/100
477/477 [==============================] - 229s 474ms/step - loss: 0.0966 -
accuracy: 0.8957 - recall: 0.8760 - val_loss: 0.1073 - val_accuracy: 0.8732 -
val_recall: 0.8480
Epoch 17/100
477/477 [==============================] - 230s 475ms/step - loss: 0.0854 -
accuracy: 0.9011 - recall: 0.8843 - val_loss: 0.0856 - val_accuracy: 0.9069 -
val_recall: 0.8852
Epoch 18/100
477/477 [==============================] - 230s 475ms/step - loss: 0.0858 -
accuracy: 0.9053 - recall: 0.8852 - val_loss: 0.0945 - val_accuracy: 0.9056 -
val_recall: 0.8816
```

```
Epoch 19/100
477/477 [==============================] - 229s 475ms/step - loss: 0.0920 -
accuracy: 0.8984 - recall: 0.8776 - val_loss: 0.0773 - val_accuracy: 0.9105 -
val_recall: 0.8954
Epoch 20/100
477/477 [==============================] - 230s 477ms/step - loss: 0.0832 -
accuracy: 0.9005 - recall: 0.8843 - val_loss: 0.1113 - val_accuracy: 0.8948 -
val_recall: 0.8828
Epoch 21/100
477/477 [==============================] - 231s 478ms/step - loss: 0.0866 -
accuracy: 0.9001 - recall: 0.8840 - val_loss: 0.0743 - val_accuracy: 0.9249 -
val_recall: 0.9087
Epoch 22/100
477/477 [==============================] - 229s 475ms/step - loss: 0.0786 -
accuracy: 0.9131 - recall: 0.8978 - val_loss: 0.0780 - val_accuracy: 0.9189 -
val_recall: 0.9093
Epoch 23/100
477/477 [==============================] - 229s 475ms/step - loss: 0.0795 -
accuracy: 0.9101 - recall: 0.8961 - val_loss: 0.0961 - val_accuracy: 0.9105 -
val_recall: 0.8924
Epoch 24/100
477/477 [==============================] - 228s 473ms/step - loss: 0.0757 -
accuracy: 0.9133 - recall: 0.9005 - val_loss: 0.0878 - val_accuracy: 0.9032 -
val_recall: 0.8948
Epoch 25/100
477/477 [==============================] - 229s 473ms/step - loss: 0.0803 -
accuracy: 0.9152 - recall: 0.9022 - val_loss: 0.1007 - val_accuracy: 0.8924 -
val_recall: 0.8786
Epoch 26/100
477/477 [==============================] - 229s 473ms/step - loss: 0.0731 -
accuracy: 0.9151 - recall: 0.9041 - val_loss: 0.0729 - val_accuracy: 0.9159 -
val_recall: 0.9032
Epoch 27/100
477/477 [==============================] - 230s 475ms/step - loss: 0.0787 -
accuracy: 0.9152 - recall: 0.9032 - val_loss: 0.0679 - val_accuracy: 0.9225 -
val_recall: 0.9111
Epoch 28/100
477/477 [==============================] - 229s 474ms/step - loss: 0.0711 -
accuracy: 0.9141 - recall: 0.9040 - val_loss: 0.0725 - val_accuracy: 0.9219 -
val_recall: 0.9129
Epoch 29/100
477/477 [==============================] - 230s 476ms/step - loss: 0.0703 -
accuracy: 0.9202 - recall: 0.9104 - val_loss: 0.0867 - val_accuracy: 0.9159 -
val_recall: 0.9038
Epoch 30/100
477/477 [==============================] - 230s 476ms/step - loss: 0.0699 -
accuracy: 0.9209 - recall: 0.9100 - val_loss: 0.0746 - val_accuracy: 0.9153 -
val_recall: 0.9081
```

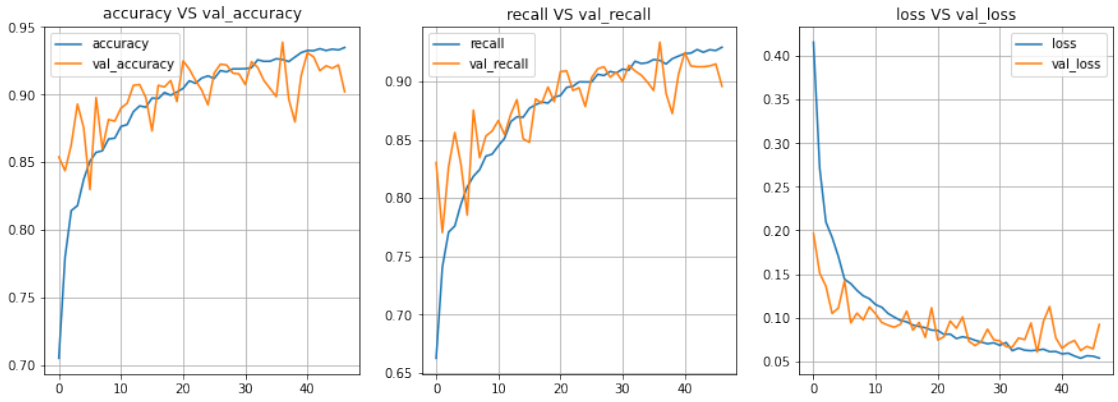
```
Epoch 31/100
477/477 [==============================] - 231s 477ms/step - loss: 0.0706 -
accuracy: 0.9145 - recall: 0.9066 - val_loss: 0.0732 - val_accuracy: 0.9075 -
val_recall: 0.9002
Epoch 32/100
477/477 [==============================] - 230s 477ms/step - loss: 0.0685 -
accuracy: 0.9240 - recall: 0.9144 - val_loss: 0.0667 - val_accuracy: 0.9243 -
val_recall: 0.9141
Epoch 33/100
477/477 [==============================] - 230s 475ms/step - loss: 0.0609 -
accuracy: 0.9270 - recall: 0.9191 - val_loss: 0.0662 - val_accuracy: 0.9201 -
val_recall: 0.9093
Epoch 34/100
477/477 [==============================] - 230s 477ms/step - loss: 0.0666 -
accuracy: 0.9256 - recall: 0.9175 - val_loss: 0.0766 - val_accuracy: 0.9105 -
val_recall: 0.9056
Epoch 35/100
477/477 [==============================] - 230s 476ms/step - loss: 0.0597 -
accuracy: 0.9272 - recall: 0.9189 - val_loss: 0.0747 - val_accuracy: 0.9044 -
val_recall: 0.8996
Epoch 36/100
477/477 [==============================] - 231s 478ms/step - loss: 0.0613 -
accuracy: 0.9264 - recall: 0.9184 - val_loss: 0.0938 - val_accuracy: 0.8984 -
val_recall: 0.8924
Epoch 37/100
477/477 [==============================] - 230s 475ms/step - loss: 0.0628 -
accuracy: 0.9275 - recall: 0.9193 - val_loss: 0.0603 - val_accuracy: 0.9387 -
val_recall: 0.9339
Epoch 38/100
477/477 [==============================] - 229s 475ms/step - loss: 0.0642 -
accuracy: 0.9255 - recall: 0.9167 - val_loss: 0.0956 - val_accuracy: 0.8966 -
val_recall: 0.8900
Epoch 39/100
477/477 [==============================] - 229s 474ms/step - loss: 0.0588 -
accuracy: 0.9289 - recall: 0.9208 - val_loss: 0.1127 - val_accuracy: 0.8798 -
val_recall: 0.8726
Epoch 40/100
477/477 [==============================] - 230s 475ms/step - loss: 0.0603 -
accuracy: 0.9315 - recall: 0.9230 - val_loss: 0.0766 - val_accuracy: 0.9141 -
val_recall: 0.9062
Epoch 41/100
477/477 [==============================] - 228s 472ms/step - loss: 0.0590 -
accuracy: 0.9298 - recall: 0.9207 - val_loss: 0.0647 - val_accuracy: 0.9309 -
val_recall: 0.9249
Epoch 42/100
477/477 [==============================] - 230s 475ms/step - loss: 0.0563 -
accuracy: 0.9359 - recall: 0.9288 - val_loss: 0.0704 - val_accuracy: 0.9279 -
val_recall: 0.9135
```

```
Epoch 43/100
477/477 [==============================] - 230s 475ms/step - loss: 0.0578 -
accuracy: 0.9296 - recall: 0.9229 - val_loss: 0.0737 - val_accuracy: 0.9177 -
val_recall: 0.9129
Epoch 44/100
477/477 [==============================] - 229s 474ms/step - loss: 0.0542 -
accuracy: 0.9338 - recall: 0.9264 - val_loss: 0.0622 - val_accuracy: 0.9213 -
val_recall: 0.9129
Epoch 45/100
477/477 [==============================] - 229s 474ms/step - loss: 0.0566 -
accuracy: 0.9338 - recall: 0.9278 - val_loss: 0.0670 - val_accuracy: 0.9195 -
val_recall: 0.9135
Epoch 46/100
477/477 [==============================] - 230s 476ms/step - loss: 0.0570 -
accuracy: 0.9327 - recall: 0.9258 - val_loss: 0.0641 - val_accuracy: 0.9219 -
val_recall: 0.9153
Epoch 47/100
477/477 [==============================] - 229s 474ms/step - loss: 0.0529 -
accuracy: 0.9348 - recall: 0.9292 - val_loss: 0.0921 - val_accuracy: 0.9020 -
val_recall: 0.8960
```

[13]:
```
transfer_model.save("./
↪tranferlearning_densenet169_with_balanced_focal_loss_3_class.h5")
```

[14]:
```
def plot_metrics(history,metrics=[]): #retorna una lista de tuplas
    fig,axes=plt.subplots(1,len(metrics))
    fig.set_size_inches(15,5)
    graph=pd.DataFrame(history)
    for i,ax in enumerate(axes.flat):
        graph[list(metrics[i])].plot(kind="line",style="-",ax=ax)
        ax.set_title(" VS ".join(list(metrics[i])))
        ax.grid(True)
    plt.show()
```

[15]:
```
metrics=[("accuracy","val_accuracy"),("recall","val_recall"),("loss","val_loss")]
plot_metrics(history_model.history,metrics=metrics)
```



[16]:
```
#obtenemos el numero de epocas donde se detuvo y lo configuramos como un epoch
↪inicial para el siguiente
#entrenamiento
EPOCH_STOP=len(history_model.epoch)
print("El modelo se entreno en",EPOCH_STOP,"Epochs")
```

```
El modelo se entreno en 47 Epochs
```

- El entrenamiento del modelo se detuvo en 47 epochs lo que nos dice que la funcion de perdida en la data de validacion no mejoro por 10 epochs consecutivos, probablemente, ya no mejore para futuras epocas.

Ahora veamos el rendimiento del modelo base en los datos de entrenamiento y validacion

[17]:
```
transfer_model.evaluate(train_generator)
```

```
477/477 [==============================] - 244s 512ms/step - loss: 0.0458 -
accuracy: 0.9466 - recall: 0.9389
```

[17]:
```
[0.04583195969462395, 0.9465808868408203, 0.938902735710144]
```

[18]:
```
transfer_model.evaluate(validation_generator)
```

```
53/53 [==============================] - 8s 146ms/step - loss: 0.0606 -
accuracy: 0.9374 - recall: 0.9327
```

[18]:
```
[0.0605820007622242, 0.9374262094497681, 0.9327036738395691]
```

- Mostramos la matriz de confusion en los datos de entrenamiento y validacion

Para el conjunto de entrenamiento

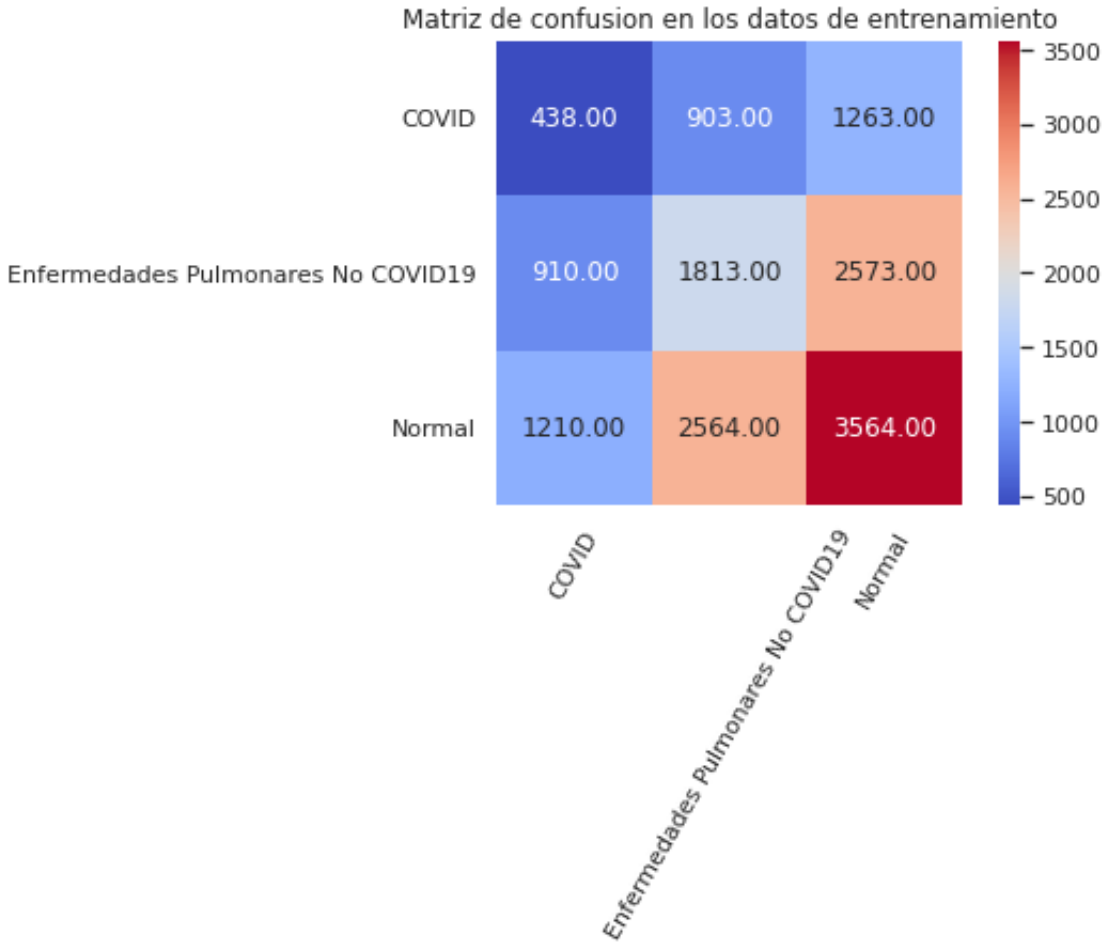
[19]:
```
y_true=train_generator.classes
predictions=transfer_model.predict(train_generator)
y_pred=np.argmax(predictions,axis=1)
```

[20]:
```
print("Indices de clase")
for idx,clase in train_generator.class_indices.items():
    print(idx,":",clase)
```

```
Indices de clase
COVID : 0
Enfermedades Pulmonares No COVID19 : 1
Normal : 2
```

[21]:
```
#obtenemos la matriz de confusion de sklearn
from sklearn.metrics import confusion_matrix
import seaborn as sns; sns.set()
classes=train_generator.class_indices.keys()
```

```
mat_train=confusion_matrix(y_true,y_pred)
sns.heatmap(mat_train,square=True,annot=True,fmt="0.
 ↪2f",cmap="coolwarm",xticklabels=classes,yticklabels=classes)
plt.xticks(rotation=60)
plt.title("Matriz de confusion en los datos de entrenamiento")
plt.show()
```



- Reporte de clasificacion para el conjunto de entrenamiento

[22]:
```
report=classification_report(y_true,y_pred,target_names=list(train_generator.
 ↪class_indices.keys()))
print(report)
```

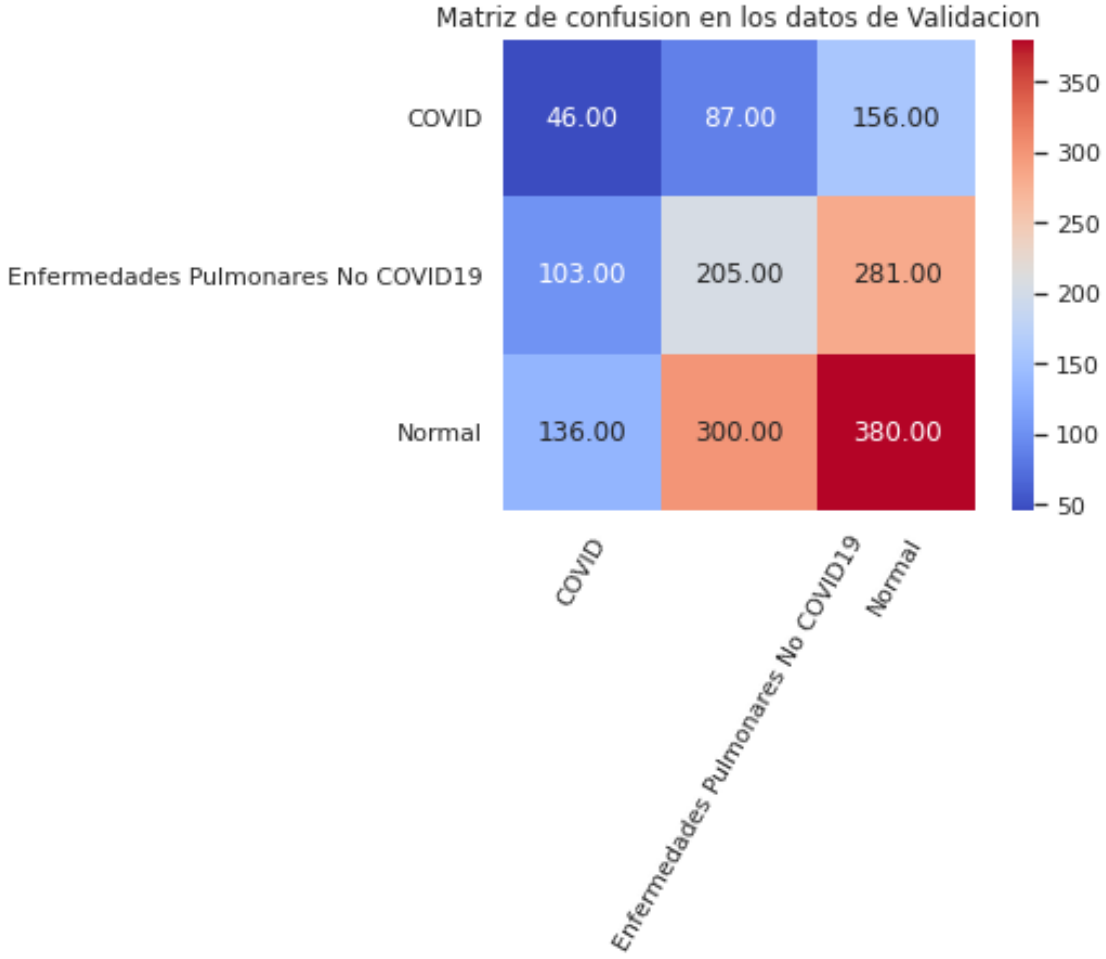
```
                                precision    recall  f1-score   support

                         COVID       0.17      0.17      0.17      2604
Enfermedades Pulmonares No COVID19       0.34      0.34      0.34      5296
                        Normal       0.48      0.49      0.48      7338
```

accuracy			0.38	15238
macro avg	0.33	0.33	0.33	15238
weighted avg	0.38	0.38	0.38	15238

- Para el conjunto de validacion

[23]:
```
y_true=validation_generator.classes
predictions=transfer_model.predict(validation_generator)
y_pred=np.argmax(predictions,axis=1)
```

[24]:
```
classes=validation_generator.class_indices.keys()
mat_val=confusion_matrix(y_true,y_pred)
sns.heatmap(mat_val,square=True,annot=True,fmt="0.
↪2f",cmap="coolwarm",xticklabels=classes,yticklabels=classes)
plt.xticks(rotation=60)
plt.title("Matriz de confusion en los datos de Validacion")
plt.show()
```



- Reporte de clasificacion para los datos de validacion

[25]:
```
report=classification_report(y_true,y_pred,target_names=list(train_generator.
↪class_indices.keys()))
print(report)
```

```
                                precision    recall  f1-score   support

                         COVID       0.16      0.16      0.16       289
Enfermedades Pulmonares No COVID19       0.35      0.35      0.35       589
                        Normal       0.47      0.47      0.47       816

                      accuracy                           0.37      1694
                     macro avg       0.32      0.32      0.32      1694
                  weighted avg       0.37      0.37      0.37      1694
```

1.0.1 RESULTADOS FINALES: MODELO TRANFER LEARNING CON PERDIDA FOCAL PARA EL BALANCEO DE CLASES. USANDO 3 CLASES

- El modelo base ha obtenido un puntaje de accuracy **ACC=94.66%** y recall **RECALL=93.89%** en el **conjunto de entrenamiento**.
- El modelo ha obtendio un puntaje de accuracy **ACC=93.74%** y recall **RECALL=93.27%** en el **conjunto de validacion**.

IMPORTANTE: El modelo ha alcanzado equilibrio entre los datos de entrenamiento y validacion, lo que siginifica que es un modelo final con aproximadamente 93% de precision. Este modelo pasará a la **Fase de evaluación del modelo**.

Evaluacion del modelo

August 29, 2021

[54]:
```
import tensorflow as tf
from sklearn.metrics import classification_report
import seaborn as sns;sns.set()
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix
import numpy as np
import tensorflow.keras as K
from skimage.transform import resize
from tensorflow.keras.models import Model
import cv2
```

0.0.1 Cargamos los mejores modelos serializados

Los mejores modelos de todos los candidatos fueron 2:

- Modelo usando transfer learning DenseNet169 con balanceo de Penalizacion de clases
- Modelo usando tranfer learning sobre DenseNet169 con balanceo de perdida focal

- Primero cargamos la funcion personalizada usada para la perdida focal, esta es importante para realizar la evaluacion del modelo

[2]:
```
def focal_loss(gamma=2., alpha=4.):

    gamma = float(gamma)
    alpha = float(alpha)

    def focal_loss_fixed(y_true, y_pred):
        """Focal loss for multi-classification
        FL(p_t)=-alpha(1-p_t)^{gamma}ln(p_t)
        Notice: y_pred is probability after softmax
        gradient is d(Fl)/d(p_t) not d(Fl)/d(x) as described in paper
        d(Fl)/d(p_t) * [p_t(1-p_t)] = d(Fl)/d(x)
        Focal Loss for Dense Object Detection
        https://arxiv.org/abs/1708.02002
        Arguments:
            y_true {tensor} -- ground truth labels, shape of [batch_size,␣
↪num_cls]
            y_pred {tensor} -- model's output, shape of [batch_size, num_cls]
```

```
        Keyword Arguments:
            gamma {float} -- (default: {2.0})
            alpha {float} -- (default: {4.0})
        Returns:
            [tensor] -- loss.
        """
        epsilon = 1.e-9
        y_true = tf.convert_to_tensor(y_true, tf.float32)
        y_pred = tf.convert_to_tensor(y_pred, tf.float32)

        model_out = tf.add(y_pred, epsilon)
        ce = tf.multiply(y_true, -tf.math.log(model_out))
        weight = tf.multiply(y_true, tf.pow(tf.subtract(1., model_out), gamma))
        fl = tf.multiply(alpha, tf.multiply(weight, ce))
        reduced_fl = tf.reduce_max(fl, axis=1)
        return tf.reduce_mean(reduced_fl)
    return focal_loss_fixed
```

[3]:
```
path_model_weighted="../input/modelosfinales/
 ↪tranferlearning_densenet169_with_balanced_focal_loss_3_class_equalized_wgt.
 ↪h5"
path_model_focal="../input/modelosfinales/
 ↪tranferlearning_densenet169_with_balanced_focal_loss_3_class_equalized.h5"
#modelo con penalizacion de clases para el balanceo
model_transfer_class_weight=tf.keras.models.load_model(path_model_weighted)
#modelo con balanceo usando perdida focal
model_transfer_focal_loss=tf.keras.models.load_model(path_model_focal,␣
 ↪custom_objects={'focal_loss_fixed': focal_loss()})
```

[4]:
```
model_transfer_class_weight.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 256, 256, 3)]     0
_________________________________________________________________
densenet169 (Functional)     (None, 8, 8, 1664)        12642880
_________________________________________________________________
flatten_1 (Flatten)          (None, 106496)            0
_________________________________________________________________
batch_normalization_2 (Batch (None, 106496)            425984
_________________________________________________________________
dense_3 (Dense)              (None, 256)               27263232
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
```

```
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 387
=================================================================
Total params: 40,366,403
Trainable params: 33,423,619
Non-trainable params: 6,942,784
_________________________________________________________________
```

[5]:
```
model_transfer_focal_loss.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 256, 256, 3)]     0
_________________________________________________________________
densenet169 (Functional)     (None, 8, 8, 1664)        12642880
_________________________________________________________________
flatten_1 (Flatten)          (None, 106496)            0
_________________________________________________________________
batch_normalization_2 (Batch (None, 106496)            425984
_________________________________________________________________
dense_3 (Dense)              (None, 256)               27263232
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 387
=================================================================
Total params: 40,366,403
Trainable params: 33,423,619
Non-trainable params: 6,942,784
_________________________________________________________________
```

0.0.2 Cargamos el flujo de datos de TEST

- A traves de la libreria

[6]:
```
path_test="../input/datasettestcovid19/test"
IMAGE_SIZE=(256,256)
test_datagen=ImageDataGenerator(rescale=1.0/255)
test_generator=test_datagen.flow_from_directory(
            path_test,
            target_size=IMAGE_SIZE,
            batch_size=32,
            shuffle=False,
            class_mode="categorical"
)
```

```
Found 4233 images belonging to 3 classes.
```

[7]:
```
test_generator.class_indices
```

[7]:
```
{'COVID': 0, 'Enfermedades Pulmonares No COVID19': 1, 'Normal': 2}
```

[8]:
```
test_generator.reset()
```

[9]:
```
model_transfer_class_weight.evaluate(test_generator)
```

```
133/133 [==============================] - 36s 201ms/step - loss: 0.2244 -
accuracy: 0.9213 - recall: 0.9178
```

[9]:
```
[0.22444237768650055, 0.9213323593139648, 0.9177888035774231]
```

[10]:
```
test_generator.reset()
```

[11]:
```
model_transfer_focal_loss.evaluate(test_generator)
```

```
133/133 [==============================] - 19s 120ms/step - loss: 0.2670 -
accuracy: 0.9242 - recall: 0.9187
```

[11]:
```
[0.26700618863105774, 0.924167275428772, 0.9187337756156921]
```

- Y efectivamente el modelo que logro una mayor precision fue el Transferencia de informacion con un accuracy de 92.42% de aciertos en datos **NUNCA ANTES VISTOS**.

Elaboremos su matriz de clasificación.

[15]:
```
test_generator.reset()
```

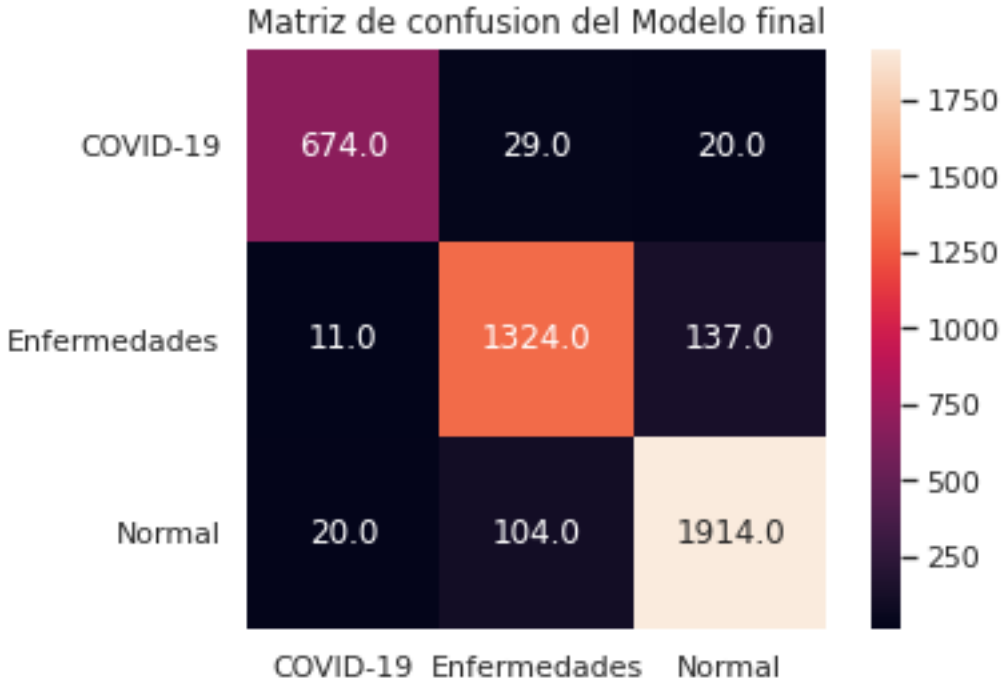
[16]:
```
y_true=test_generator.classes
predictions=model_transfer_focal_loss.predict(test_generator)
y_pred=np.argmax(predictions,axis=1)
```

[20]:
```
test_generator.class_indices
```

[20]:
```
{'COVID': 0, 'Enfermedades Pulmonares No COVID19': 1, 'Normal': 2}
```

Matriz de confusion En esta tendremos una version mas realista de cada clase mal predecida.

[23]:
```
target_names={"COVID-19":0,"Enfermedades":1,"Normal":2}#test_generator.
 ↪class_indices
mat=confusion_matrix(y_true,y_pred)


sns.heatmap(mat,annot=True,fmt="0.
 ↪1f",square=True,xticklabels=target_names,yticklabels=target_names)
plt.title("Matriz de confusion del Modelo final")
plt.show()
```



- **Reporte de clasificacion**: Aqui tendremos un reporte sobre la precision en las predicciones de cada clase.

[24]:
```
report=classification_report(y_true,y_pred,target_names=target_names)
print(report)
```

```
              precision    recall  f1-score   support

    COVID-19       0.96      0.93      0.94       723
Enfermedades       0.91      0.90      0.90      1472
      Normal       0.92      0.94      0.93      2038

    accuracy                           0.92      4233
```

```
   macro avg       0.93      0.92      0.93      4233
weighted avg       0.92      0.92      0.92      4233
```

0.1 INTERPRETABILIDAD DEL MODELO

- A veces la pregunta suele ser porque modelo predice lo que predice, para ello usaremos el metodo de activacion en capas llamado CamGrad que nos muestra un mapa de calor sobre las partes donde el modelo ha tenido mayor actividad sobre la imagen a clasificar, esto nos puede ayudar a determinar que lugares de una radiografia son determinantes para predecir si una radiografia de rayos-X es de COVID-19

El codigo usado en el siguiente espacio pertenece a la libreria Keras

[3]:
```
path_model_focal="../Models/
↪tranferlearning_densenet169_with_balanced_focal_loss_3_class_equalized.h5"
model_transfer_focal_loss=tf.keras.models.load_model(path_model_focal,␣
↪custom_objects={'focal_loss_fixed': focal_loss()})
```

[31]:
```
```

[34]:
```
model_transfer_focal_loss.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 256, 256, 3)]     0
_________________________________________________________________
densenet169 (Functional)     (None, 8, 8, 1664)        12642880
_________________________________________________________________
flatten_1 (Flatten)          (None, 106496)            0
_________________________________________________________________
batch_normalization_2 (Batch (None, 106496)            425984
_________________________________________________________________
dense_3 (Dense)              (None, 256)               27263232
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 387
=================================================================
Total params: 40,366,403
Trainable params: 33,423,619
```

```
Non-trainable params: 6,942,784
_________________________________________________________________
```

- Una vez obtenido el modelo de transfer learning. Necesitamos la imagen preprocesada

[24]:
```
tf.keras.Model
```

[24]:
```
tensorflow.python.keras.engine.training.Model
```

[83]:
```
import random
import os
def get_random_path(origin):
    return random.sample([ arch.path for arch in os.scandir(origin)],1)[0]
def preprocess_image(img_path,target_size=None):
    img = tf.keras.preprocessing.image.load_img(img_path,␣
 ↪target_size=target_size)
    array = tf.keras.preprocessing.image.img_to_array(img)
    #array=np.expand_dims(array,axis=0)
    array=array*1.0/255
    return array

img_path=get_random_path("../Datasets/test/COVID")
img_array=preprocess_image(img_path,target_size=(256,256))
```

[90]:
```
print("La imagen fue extraida de:",img_path)
```

```
La imagen fue extraida de: ../Datasets/test/COVID\COVID-103.png
```

Implementacion de algoritmo de GradCAM

[53]:
```
#aqui obtenemos el modelo original del transfer learning
def get_transfer_model(model):
    model_transfer=None
    for layer in model.layers:
        if isinstance(layer,tf.keras.Model):
            model_transfer=layer
            break
    return model_transfer

def get_last_conv_layer(model):
    for layer in model.layers[::-1]:
        if isinstance(layer,K.layers.Conv2D):
            return layer
    return None
```

[97]:
```
def VizGradCAM(model, image, interpolant=0.6, return_gradcam=True):

    # Sanity Check
```

```
    #if interpolant < 0 or interpolant >1:
    #    print("Heatmap Interpolation Must Be Between 0 - 1")
    #   return None

    transfer_model=get_transfer_model(model)

    last_conv_layer = get_last_conv_layer(transfer_model)

    target_layer = transfer_model.get_layer(last_conv_layer.name)

    original_img = image
    img = np.expand_dims(original_img, axis=0)
    prediction = model.predict(img)

    # Obtain Prediction Index
    prediction_idx = np.argmax(prediction)

    # Compute Gradient of Top Predicted Class
    with tf.GradientTape() as tape:
        gradient_model = Model([transfer_model.inputs], [target_layer.output,␣
↪transfer_model.output])
        conv2d_out, prediction = gradient_model(img)
        # Obtain the Prediction Loss
        loss = prediction[:, prediction_idx]

    # Gradient() computes the gradient using operations recorded
    # in context of this tape
    gradients = tape.gradient(loss, conv2d_out)

    # Obtain the Output from Shape [1 x H x W x CHANNEL] -> [H x W x CHANNEL]
    output = conv2d_out[0]

    # Obtain Depthwise Mean
    weights = tf.reduce_mean(gradients[0], axis=(0, 1))

    # Create a 7x7 Map for Aggregation
    activation_map = np.zeros(output.shape[0:2], dtype=np.float32)

    # Multiply Weights with Every Layer
    for idx, weight in enumerate(weights):
        activation_map += weight * output[:, :, idx]

    # Resize to Size of Image
    activation_map = cv2.resize(
        activation_map.numpy(), (original_img.shape[1], original_img.shape[0])
    )
```

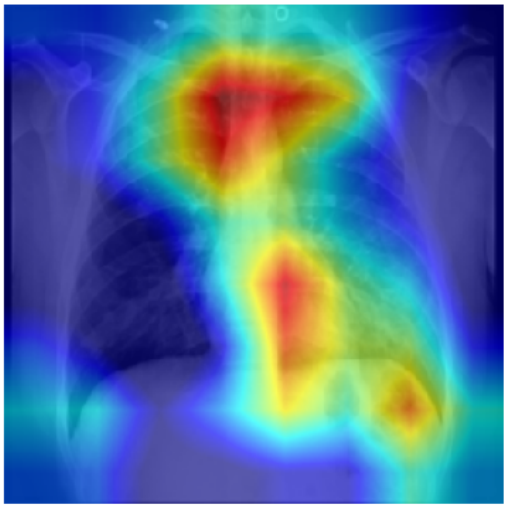
```
    # Ensure No Negative Numbers
    activation_map = np.maximum(activation_map, 0)

    # Convert Class Activation Map to 0 - 255
    activation_map = (activation_map - activation_map.min()) / (
        activation_map.max() - activation_map.min()
    )
    activation_map = np.uint8(255 * activation_map)

    # Convert to Heatmap
    heatmap = cv2.applyColorMap(activation_map, cv2.COLORMAP_JET)

    # Superimpose Heatmap on Image Data
    original_img = np.uint8(
        (original_img - original_img.min())
        / (original_img.max() - original_img.min())
        * 255
    )

    cvt_heatmap = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB)

    # Enlarge Plot
    plt.rcParams["figure.dpi"] = 100

    if return_gradcam == True:
        return np.uint8(original_img * interpolant + cvt_heatmap * (1 -␣
↪interpolant))

        #plt.savefig("./grad_cam_image.png")
    else:
        return cvt_heatmap
```

[117]:
```
def compare_gradcam(model,path_image):
    fig,(ax1,ax2)=plt.subplots(1,2)
    fig.set_size_inches(10,10)
    array_image=preprocess_image(path_image,target_size=(256,256))
    im=plt.imread(path_image)
    ax1.imshow(im,cmap="bone")
    ax1.set_title("Radiografia con COVID-19")
    ax1.axis(False)
    array_gradcam=VizGradCAM(model, img_array, return_gradcam=True,␣
↪interpolant=0.35)
    ax2.imshow(array_gradcam)
    ax2.set_title("Mayor actividad del Modelo")
    ax2.axis(False)
```

[118]:
```
compare_gradcam(model_transfer_focal_loss,img_path)
```

Radiografia con COVID-19

Mayor actividad del Modelo



- Observamos que el modelo tiene mayor actividad sobre zonas centrales en los pulmones, esto indica cierto factor determinante para escoger COVID-19

0.2 CONCLUSIONES:

Este modelo posee una precision del 92% sobre datos nunca antes vistos, a su vez podemos afirmar que clasifica correctamente como:

- COVID-19 el 96% de los casos
- Enfermedades pulmonares el 91% de los casos
- Radiografias normales el 92% de los casos

El siguiente paso en la metodologia CRISP-DM es el despliegue del modelo a produccion.

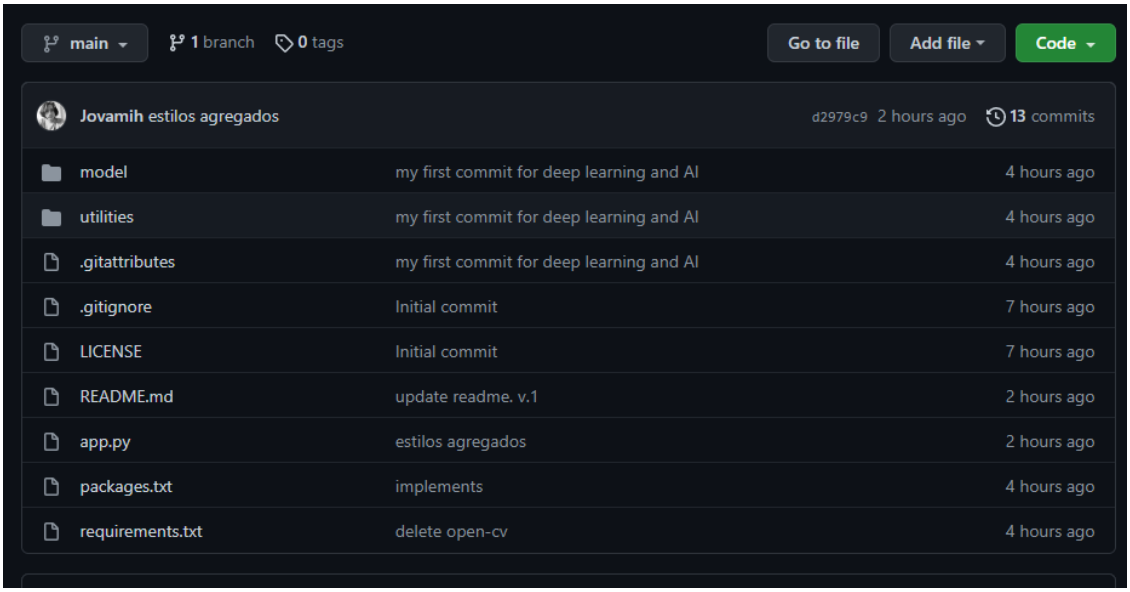
Despliegue del Modelo

August 29, 2021

- Para que el modelo sea de utilidad para los organizaciones y para poder probarla para demostrar su eficacia, desplegamos el modelo a produccion. Para ello evaluamos las siguientes plataformas en la nube tomando en cuenta **CRITERIOS DE SERVICIOS DE DISPONIBILIDAD GRATUITA PARA APLICACIONES WEB** para el despliegue de modelos a producción.

Plataforma	Modo gratuito	Servicio	Limite de peso de archivo	Disponibilidad limite	Viable	Observacion
Azure	SI	Azure Web Apps	1GB	1 Hora de CPU/Dia	NO	
GCP	SI	TFX	1GB	3 Meses de disponibilidad gratuita	NO	
AWS	SI	Elastic Beanstack	512MB	Free/ilimitada	SI	La implementacion dificil
Heroku	SI	Create App	512MB	Free/ilimitada	SI	El modelo pesaba mucho
Streamlit Sharing	**SI**	**Deploy app**	**null**	**Free/ilimitada**	**SI**	**Plataforma escogida**

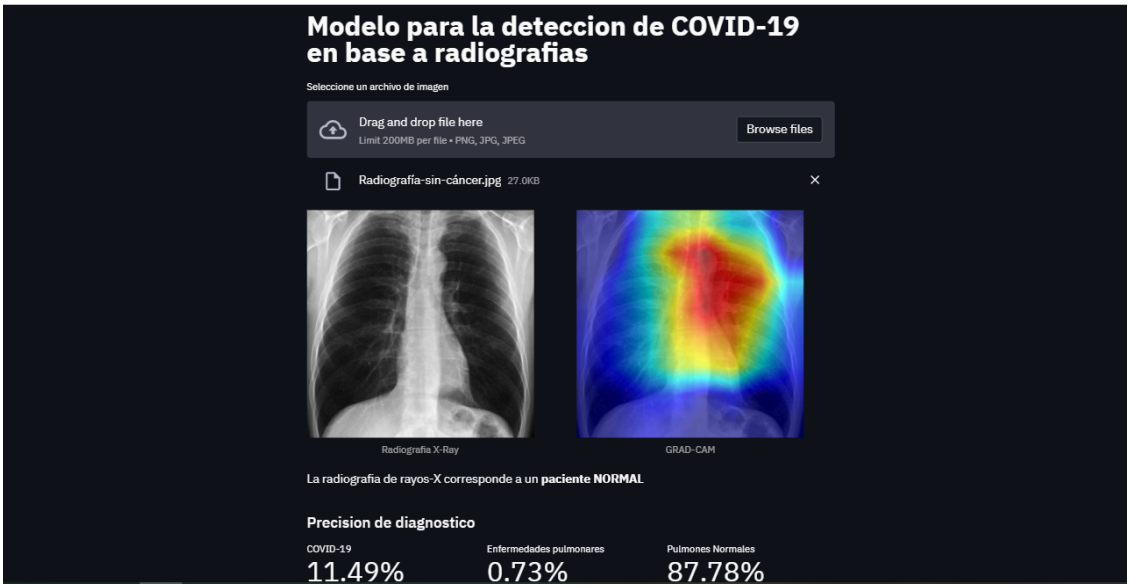
- Una vez escogida la plataforma procedemos a implementar la aplicacion a traves de los marcos de trabajo de streamlit, una plataforma para compartir aplicaciones de ciencia de datos de manera sencilla.



- El link de mi repositorio: **https://github.com/Jovamih/Covid19DetectorApp**

Y por ultimo y no menos importante algunas indicaciones para usar la aplicacion.

1. Cargar la imagen de la radiografia de pulmones.
2. Visualizar los resultados asociados, tanto como la probabilidad de diagnostico de cada enfermedad como tambien el verdicto final.
3. ... Si quieres mas funcionalidades y posibles mejoras para el modelo, sugiero mandarme un feedback



0.1 Acerca del autor:

0.1.1 Johan valerio Mitma Huaccha. 20 años. Perú.

- En dedicatoria a mi madre. **Isabel Huaccha Fernandez**.

Feliz ciencia de datos, feliz machine learning. Gracias por llegar al final de este laborioso trabajo de ciencia de datos.