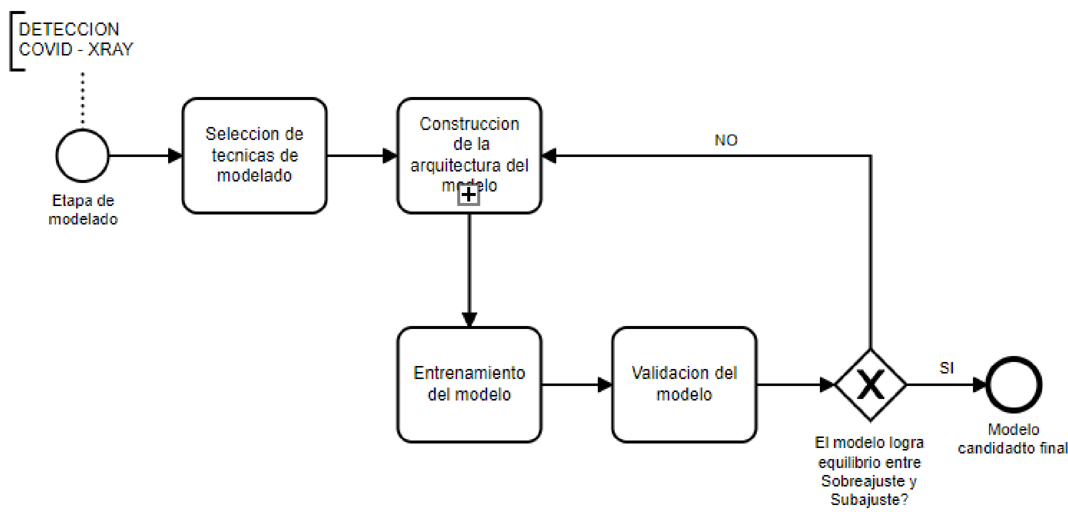
Modelado usando Transfer Learning con DenseNet169 y balanceo por Perdida Focal

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()
```

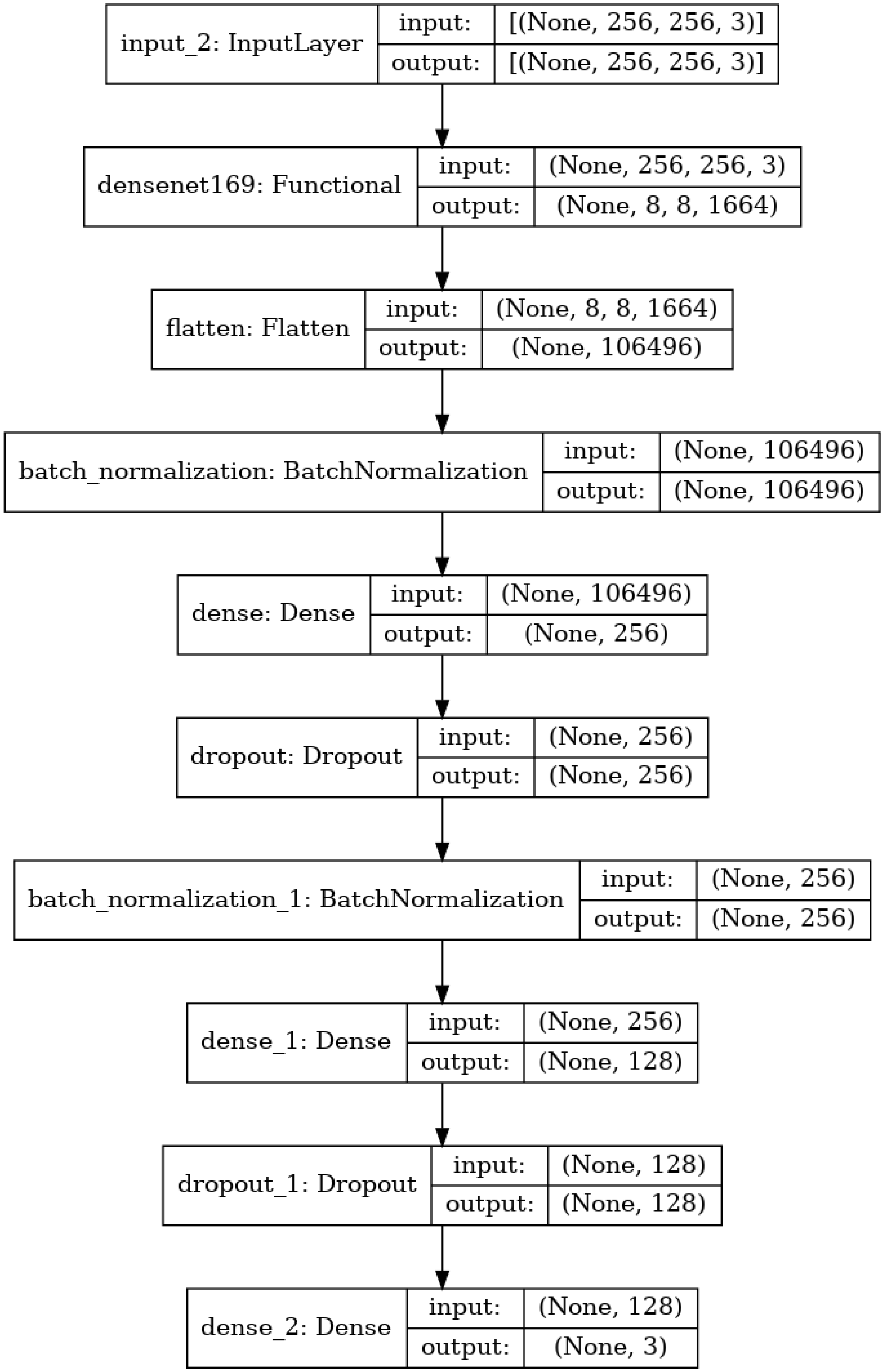
[10]:
```
transfer_model.summary()
```

```
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
```

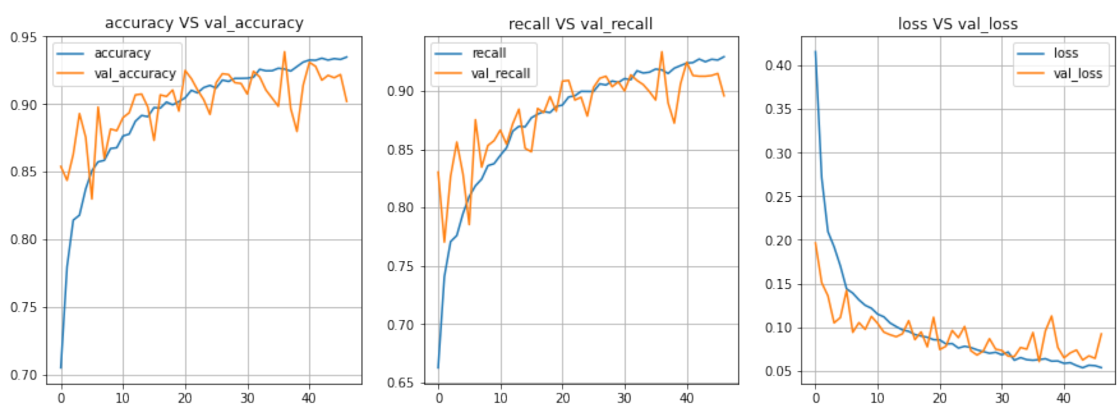
```
Epoch 31/100
477/477 [==============================] - 231s 477ms/step - loss: 0.0706 -
accuracy: 0.9145 - recall: 0.9066 - val_loss: 0.0732 - val_accuracy: 0.9075 -
val_recall: 0.9002
Epoch 32/100
477/477 [==============================] - 230s 477ms/step - loss: 0.0685 -
accuracy: 0.9240 - recall: 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

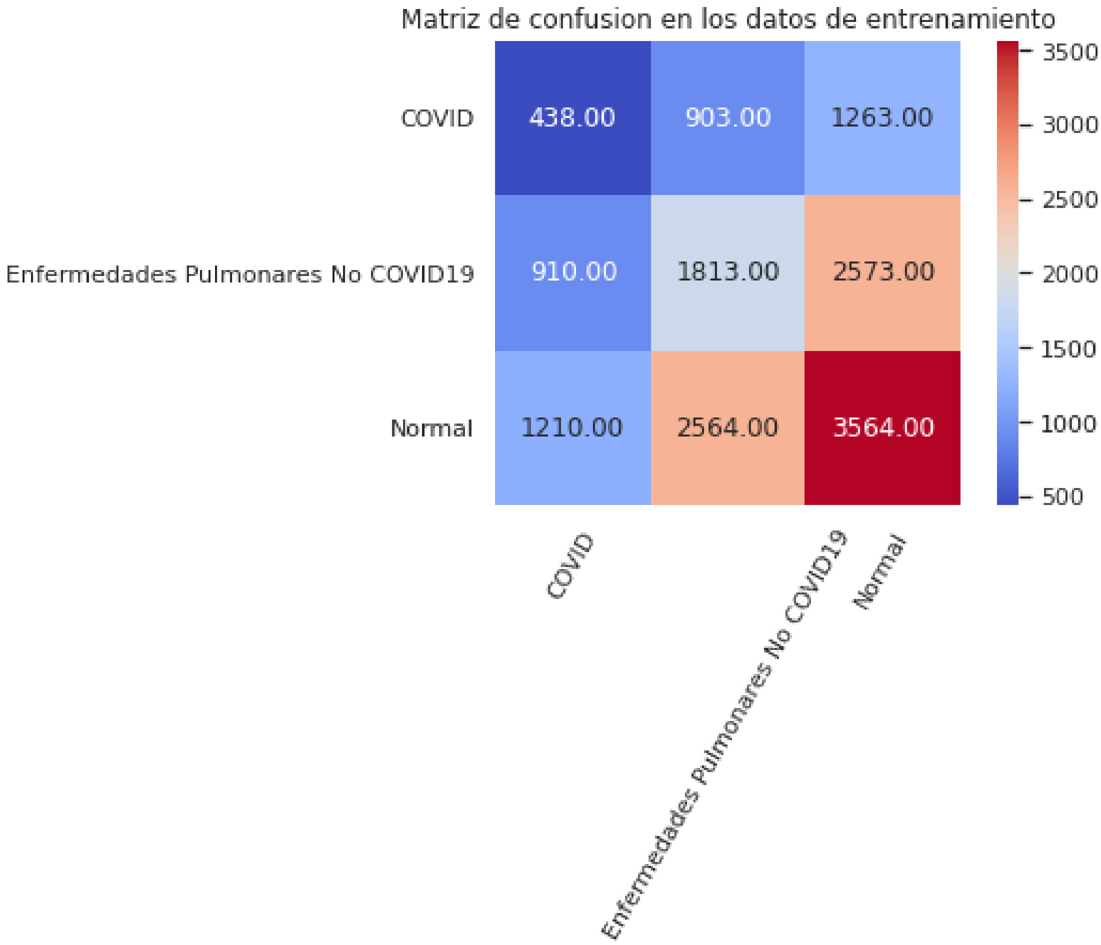
[19]:
```
y_true=train_generator.classes
predictions=transfer_model.predict(train_generator)
y_pred=np.argmax(predictions,axis=1)
```

[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)
```

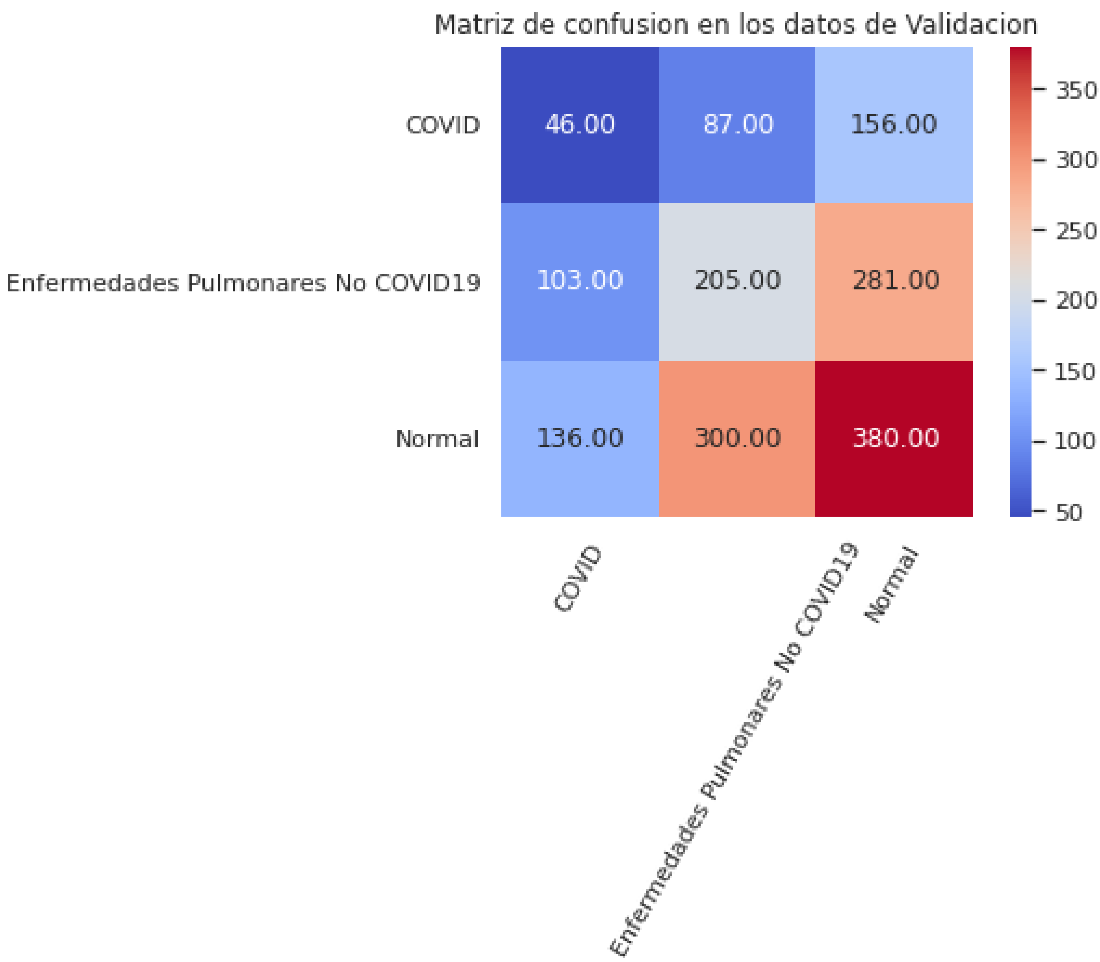
```
                                    precision    recall  f1-score   support

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

[24]:
```python
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**.