CIDANet

July 20, 2021

1 CIDANet

1.1 SplitFolders

Librería utilizada para separar las imagenes en test, train y val

[34]:
```
!pip install split-folders tdm
!pip install split-folders
```

```
Requirement already satisfied: split-folders in e:\programs\anaconda\lib\site-
packages (0.4.3)
Requirement already satisfied: tdm in e:\programs\anaconda\lib\site-packages
(0.1.0)
Requirement already satisfied: split-folders in e:\programs\anaconda\lib\site-
packages (0.4.3)
```

[46]:
```
import splitfolders
splitfolders.ratio('TFMPHOTOSCOLORED', output="CovidSplitColored", seed=1337,␣
 ↪ratio=(.8, 0.1,0.1))
```

```
Copying files: 10367 files [01:03, 163.72 files/s]
```

1.2 Inicialización del proyecto

Realizamos una serie de imports para el proyecto y definimos los espacios de trabajo donde se encuentran las imágenes.

[15]:
```
#Some Basic Imports
import matplotlib.pyplot as plt #For Visualization
import numpy as np              #For handling arrays
import pandas as pd             # For handling data
#Define Directories for train, test Set

TRAIN_PATH = "CovidSplitColored/train"
TEST_PATH = "CovidSplitColored/test"
VALID_PATH = "CovidSplitColored/val"

#Define some often used standard parameters
#The batch refers to the number of training examples utilized in one #iteration
batch_size = 16
```

```
#The dimension of the images we are going to define is 500x500
img_height = 224
img_width = 224
```

1.3 Preparación de los datos

1.3.1 Data Augmentation

Definimos las técnicas de data augmentation, solo para train.

[2]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Create Image Data Generator for Train Set
image_gen = ImageDataGenerator(
                                  rescale = 1./255,
                                  shear_range = 0.2,
                                  zoom_range = 0.2,
                                  horizontal_flip = True,
                               )
# Create Image Data Generator for Test/Validation Set
test_data_gen = ImageDataGenerator(rescale = 1./255)
```

1.3.2 Carga de las imagenes

Realizamos la carga de imágenes en batch de tamaño 16.

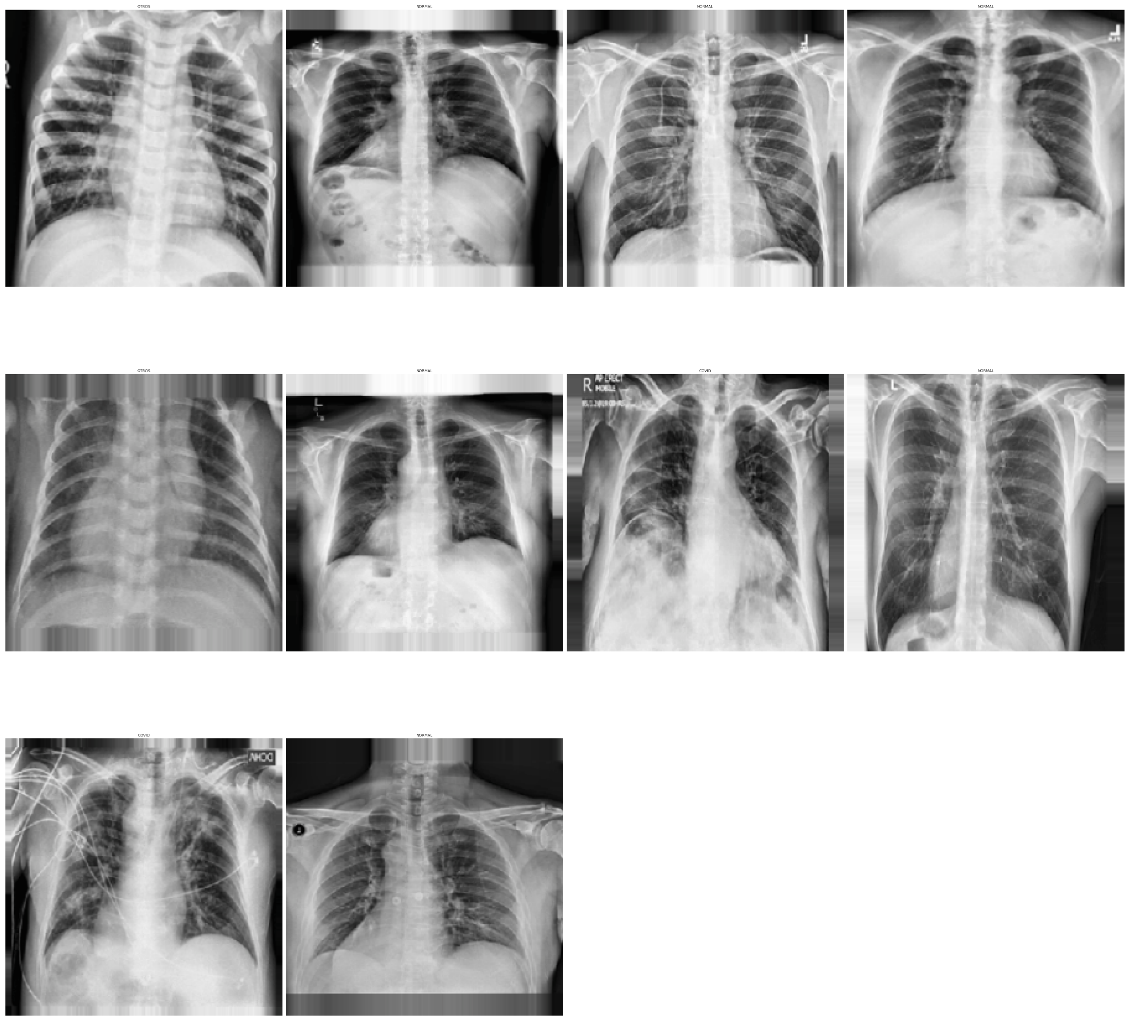
[3]:
```
train = image_gen.flow_from_directory(
      TRAIN_PATH,
      target_size=(img_height, img_width),
#       color_mode='grayscale',
      class_mode='categorical',
      batch_size=batch_size
      )
test = test_data_gen.flow_from_directory(
      TEST_PATH,
      target_size=(img_height, img_width),
#       color_mode='grayscale',
      shuffle=False,
#setting shuffle as False just so we can later compare it with predicted values␣
 ↪without having indexing problem
      class_mode='categorical',
      batch_size=batch_size
      )
valid = test_data_gen.flow_from_directory(
      VALID_PATH,
      target_size=(img_height, img_width),
#       color_mode='grayscale',
      class_mode='categorical',
      batch_size=batch_size
```

```
)
```

```
Found 8293 images belonging to 3 classes.
Found 1039 images belonging to 3 classes.
Found 1035 images belonging to 3 classes.
```

Mostramos como quedarían las imágenes, junto con sus etiquetas.

[9]:
```
plt.figure(figsize=(50, 50))
for i in range(0, 10):
    plt.subplot(3, 4, i+1)
    for X_batch, Y_batch in train:
        image = X_batch[0]
        dic = {0:'COVID',1 : 'NORMAL', 2:'OTROS'}
        plt.title(dic.get(Y_batch[0].tolist().index(1)))
        plt.axis('off')
        plt.imshow(np.squeeze(image),cmap='gray',interpolation='nearest')
        break
plt.tight_layout()
plt.show()
```



1.4 Creación de la CNN

[4]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv2D,Flatten,MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping,ReduceLROnPlateau,ModelCheckpoint
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.mobilenet import MobileNet
```

1.5 Modelos preentrenados

1.6 VGG16

[4]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

[124]:
```
from tensorflow.keras.applications import VGG16
```

Se cargan los modelos con los pesos de ImageNet.

[125]:
```
baseModel = VGG16(weights="imagenet", include_top=False,␣
 ↪input_shape=(img_height, img_width, 3))
```

Le cargamos algunas capas adicionales para adaptarlo a nuestro modelo.

[126]:
```
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64 , activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(3, activation="softmax")(headModel)
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False
```

[127]:
```
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =␣
 ↪['accuracy'])
```

[130]:
```
early_VGG16 = EarlyStopping(monitor='val_loss', mode='min', patience=3)
learning_rate_reduction_VGG16 = ReduceLROnPlateau(monitor='val_loss', patience␣
 ↪= 2, verbose=1,factor=0.3, min_lr=0.000001)
checkpoint_VGG16 = ModelCheckpoint(filepath= "CheckPoints/VGG16/
 ↪modelVGG16_{epoch:02d}.pb", save_best_only=True)

callbacks_list_VGG16 = [ early_VGG16,␣
 ↪learning_rate_reduction_VGG16,checkpoint_VGG16]
```

Como no se trata de clases equilibradas, es necesario computar los pesos de las clases para que se tengan en cuenta en el entrenamiento.

[131]:
```
from sklearn.utils.class_weight import compute_class_weight
weights = compute_class_weight('balanced', np.unique(train.classes), train.
 ↪classes)
cw = dict(zip( np.unique(train.classes), weights))
print(cw)
```

```
{0: 0.9389671361502347, 1: 0.6394884092725819, 2: 2.6936026936026938}
```

[132]:
```
model.fit(train,epochs=10, validation_data=valid, class_weight=cw,␣
 ↪callbacks=callbacks_list_VGG16)
```

```
Epoch 1/10
50/50 [==============================] - 119s 2s/step - loss: 1.0811 - accuracy:
0.4638 - val_loss: 0.9382 - val_accuracy: 0.6700
INFO:tensorflow:Assets written to: CheckPoints2/VGG16\modelVGG16_01.pb\assets
Epoch 2/10
50/50 [==============================] - 123s 2s/step - loss: 0.9106 - accuracy:
0.5888 - val_loss: 0.8346 - val_accuracy: 0.7100
INFO:tensorflow:Assets written to: CheckPoints2/VGG16\modelVGG16_02.pb\assets
Epoch 3/10
50/50 [==============================] - 122s 2s/step - loss: 0.8342 - accuracy:
0.6275 - val_loss: 0.7476 - val_accuracy: 0.7000
INFO:tensorflow:Assets written to: CheckPoints2/VGG16\modelVGG16_03.pb\assets
Epoch 4/10
50/50 [==============================] - 119s 2s/step - loss: 0.7765 - accuracy:
0.6725 - val_loss: 0.6554 - val_accuracy: 0.8000
INFO:tensorflow:Assets written to: CheckPoints2/VGG16\modelVGG16_04.pb\assets
Epoch 5/10
50/50 [==============================] - 118s 2s/step - loss: 0.7304 - accuracy:
0.7000 - val_loss: 0.6308 - val_accuracy: 0.8000
INFO:tensorflow:Assets written to: CheckPoints2/VGG16\modelVGG16_05.pb\assets
Epoch 6/10
50/50 [==============================] - 112s 2s/step - loss: 0.6816 - accuracy:
0.6950 - val_loss: 0.5831 - val_accuracy: 0.7900
INFO:tensorflow:Assets written to: CheckPoints2/VGG16\modelVGG16_06.pb\assets
Epoch 7/10
50/50 [==============================] - 118s 2s/step - loss: 0.6382 - accuracy:
0.7262 - val_loss: 0.5502 - val_accuracy: 0.8000
INFO:tensorflow:Assets written to: CheckPoints2/VGG16\modelVGG16_07.pb\assets
Epoch 8/10
50/50 [==============================] - 119s 2s/step - loss: 0.6120 - accuracy:
0.7188 - val_loss: 0.5341 - val_accuracy: 0.8000
INFO:tensorflow:Assets written to: CheckPoints2/VGG16\modelVGG16_08.pb\assets
Epoch 9/10
50/50 [==============================] - 116s 2s/step - loss: 0.5869 - accuracy:
0.7375 - val_loss: 0.5086 - val_accuracy: 0.7700
```

```
INFO:tensorflow:Assets written to: CheckPoints2/VGG16\modelVGG16_09.pb\assets
Epoch 10/10
50/50 [==============================] - 113s 2s/step - loss: 0.5961 - accuracy:
0.7325 - val_loss: 0.4786 - val_accuracy: 0.7800
INFO:tensorflow:Assets written to: CheckPoints2/VGG16\modelVGG16_10.pb\assets
```

[132]: <tensorflow.python.keras.callbacks.History at 0x1da4736ce80>

1.7 MobilenetV3

[51]:
```
# cargamos el modelo base
baseModel = MobileNet(input_shape=(img_height, img_width, 3), alpha=1,␣
↪include_top=False,
                                                       pooling='avg',␣
↪weights='imagenet')
```

[52]:
```
headModel = baseModel.output
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(256, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(3, activation="softmax")(headModel)
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False
```

[53]:
```
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =␣
↪['accuracy'])
```

[54]:
```
early_MobileNet = EarlyStopping(monitor='val_loss', mode='min', patience=7)
learning_rate_reduction_MobileNet = ReduceLROnPlateau(monitor='val_loss',␣
↪patience = 2, verbose=1,factor=0.3, min_lr=0.000001)
checkpoint_MobileNet = ModelCheckpoint(filepath= "CheckPointsColored/MobileNet/
↪modelMobileNet_{epoch:02d}.pb", save_best_only=True)

callbacks_list_MobileNet = [ early_MobileNet,␣
↪learning_rate_reduction_MobileNet,checkpoint_MobileNet]
```

[56]:
```
from sklearn.utils.class_weight import compute_class_weight
weights = compute_class_weight('balanced', np.unique(train.classes), train.
↪classes)
cw = dict(zip( np.unique(train.classes), weights))
print(cw)
```

```
{0: 0.9270064833445115, 1: 0.6527351436442346, 2: 2.56908302354399}
```

[68]:
```
model.fit(train,epochs=50, validation_data=valid, class_weight=cw,␣
↪callbacks=callbacks_list_MobileNet)
```

```
Epoch 1/50
519/519 [==============================] - 224s 432ms/step - loss: 0.1083 -
accuracy: 0.9497 - val_loss: 0.1055 - val_accuracy: 0.9614
Epoch 2/50
519/519 [==============================] - 234s 450ms/step - loss: 0.1034 -
accuracy: 0.9496 - val_loss: 0.1056 - val_accuracy: 0.9614
Epoch 3/50
519/519 [==============================] - 220s 423ms/step - loss: 0.1053 -
accuracy: 0.9509 - val_loss: 0.1057 - val_accuracy: 0.9614
Epoch 4/50
519/519 [==============================] - 219s 421ms/step - loss: 0.1060 -
accuracy: 0.9497 - val_loss: 0.1057 - val_accuracy: 0.9614
Epoch 5/50
519/519 [==============================] - 206s 396ms/step - loss: 0.1058 -
accuracy: 0.9491 - val_loss: 0.1056 - val_accuracy: 0.9604
Epoch 6/50
519/519 [==============================] - 203s 391ms/step - loss: 0.1123 -
accuracy: 0.9501 - val_loss: 0.1057 - val_accuracy: 0.9614
Epoch 7/50
519/519 [==============================] - 203s 390ms/step - loss: 0.1057 -
accuracy: 0.9529 - val_loss: 0.1054 - val_accuracy: 0.9614
Epoch 8/50
519/519 [==============================] - 203s 392ms/step - loss: 0.1009 -
accuracy: 0.9518 - val_loss: 0.1053 - val_accuracy: 0.9614
Epoch 9/50
519/519 [==============================] - 202s 388ms/step - loss: 0.0999 -
accuracy: 0.9536 - val_loss: 0.1051 - val_accuracy: 0.9614
Epoch 10/50
519/519 [==============================] - 202s 390ms/step - loss: 0.1015 -
accuracy: 0.9519 - val_loss: 0.1050 - val_accuracy: 0.9614
Epoch 11/50
519/519 [==============================] - 205s 394ms/step - loss: 0.1046 -
accuracy: 0.9510 - val_loss: 0.1050 - val_accuracy: 0.9623
Epoch 12/50
519/519 [==============================] - 203s 392ms/step - loss: 0.1041 -
accuracy: 0.9532 - val_loss: 0.1049 - val_accuracy: 0.9614
Epoch 13/50
519/519 [==============================] - 204s 392ms/step - loss: 0.1099 -
accuracy: 0.9502 - val_loss: 0.1049 - val_accuracy: 0.9614
Epoch 14/50
519/519 [==============================] - 204s 394ms/step - loss: 0.1078 -
accuracy: 0.9535 - val_loss: 0.1048 - val_accuracy: 0.9614
Epoch 15/50
519/519 [==============================] - 204s 392ms/step - loss: 0.1060 -
accuracy: 0.9500 - val_loss: 0.1049 - val_accuracy: 0.9623
```

```
Epoch 16/50
519/519 [==============================] - 204s 392ms/step - loss: 0.1077 -
accuracy: 0.9488 - val_loss: 0.1048 - val_accuracy: 0.9614
Epoch 17/50
519/519 [==============================] - 207s 399ms/step - loss: 0.1048 -
accuracy: 0.9515 - val_loss: 0.1047 - val_accuracy: 0.9614
Epoch 18/50
519/519 [==============================] - 203s 392ms/step - loss: 0.1069 -
accuracy: 0.9508 - val_loss: 0.1046 - val_accuracy: 0.9623
Epoch 19/50
519/519 [==============================] - 203s 390ms/step - loss: 0.1059 -
accuracy: 0.9519 - val_loss: 0.1046 - val_accuracy: 0.9623
Epoch 20/50
519/519 [==============================] - 203s 391ms/step - loss: 0.1009 -
accuracy: 0.9504 - val_loss: 0.1045 - val_accuracy: 0.9623
Epoch 21/50
519/519 [==============================] - 203s 391ms/step - loss: 0.1043 -
accuracy: 0.9497 - val_loss: 0.1046 - val_accuracy: 0.9614
Epoch 22/50
519/519 [==============================] - 201s 388ms/step - loss: 0.1018 -
accuracy: 0.9526 - val_loss: 0.1045 - val_accuracy: 0.9604
Epoch 23/50
519/519 [==============================] - 203s 391ms/step - loss: 0.1037 -
accuracy: 0.9503 - val_loss: 0.1044 - val_accuracy: 0.9604
Epoch 24/50
519/519 [==============================] - 202s 390ms/step - loss: 0.1074 -
accuracy: 0.9495 - val_loss: 0.1045 - val_accuracy: 0.9604
Epoch 25/50
519/519 [==============================] - 202s 389ms/step - loss: 0.1105 -
accuracy: 0.9484 - val_loss: 0.1046 - val_accuracy: 0.9604
Epoch 26/50
519/519 [==============================] - 202s 390ms/step - loss: 0.1035 -
accuracy: 0.9514 - val_loss: 0.1046 - val_accuracy: 0.9604
Epoch 27/50
519/519 [==============================] - 202s 389ms/step - loss: 0.1096 -
accuracy: 0.9466 - val_loss: 0.1049 - val_accuracy: 0.9594
Epoch 28/50
519/519 [==============================] - 202s 390ms/step - loss: 0.1070 -
accuracy: 0.9490 - val_loss: 0.1048 - val_accuracy: 0.9604
Epoch 29/50
519/519 [==============================] - 203s 392ms/step - loss: 0.1036 -
accuracy: 0.9532 - val_loss: 0.1048 - val_accuracy: 0.9594
Epoch 30/50
519/519 [==============================] - 203s 391ms/step - loss: 0.1050 -
accuracy: 0.9492 - val_loss: 0.1050 - val_accuracy: 0.9604
```

```
[68]: <tensorflow.python.keras.callbacks.History at 0x1308dc526d0>
```

1.7.1 Inception v3

[211]:
```
# cargamos el modelo base
baseModel = InceptionV3(input_shape=(img_height, img_width, 3),␣
↪include_top=False,
                                                    pooling='avg',␣
↪weights='imagenet',classifier_activation='softmax')

# y congelamos el entrenamiento en todas las capas
for layer in inception.layers:
    layer.trainable = False
```

[212]:
```
headModel = baseModel.output
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(256, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(3, activation="softmax")(headModel)
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False
```

[213]:
```
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =␣
↪['accuracy'])
```

[216]:
```
early_inception = EarlyStopping(monitor='val_loss', mode='min', patience=3)
learning_rate_reduction_inception = ReduceLROnPlateau(monitor='val_loss',␣
↪patience = 2, verbose=1,factor=0.3, min_lr=0.000001)
checkpoint_inception = ModelCheckpoint(filepath= "CheckPoints/InceptionFINAL/
↪modelInception_{epoch:02d}.pb", save_best_only=True)

callbacks_list_inception = [ early_inception,␣
↪learning_rate_reduction_inception,checkpoint_inception]
```

[218]:
```
from sklearn.utils.class_weight import compute_class_weight
weights = compute_class_weight('balanced', np.unique(train.classes), train.
↪classes)
cw = dict(zip( np.unique(train.classes), weights))
print(cw)
```

```
{0: 0.9454817888427847, 1: 0.6456513183785911, 2: 2.541201982651797}
```

[ ]:
```
model.fit(train,epochs=20, validation_data=valid, class_weight=cw,␣
↪callbacks=callbacks_list_inception)
```

1.8 Cargar Modelo

Gracias a los CallBacks definidos, podemos cargar el modelo que queramos en cualquier momento.

[1]:
```
from tensorflow.keras.models import load_model
```

[2]:
```
import tensorflow
```

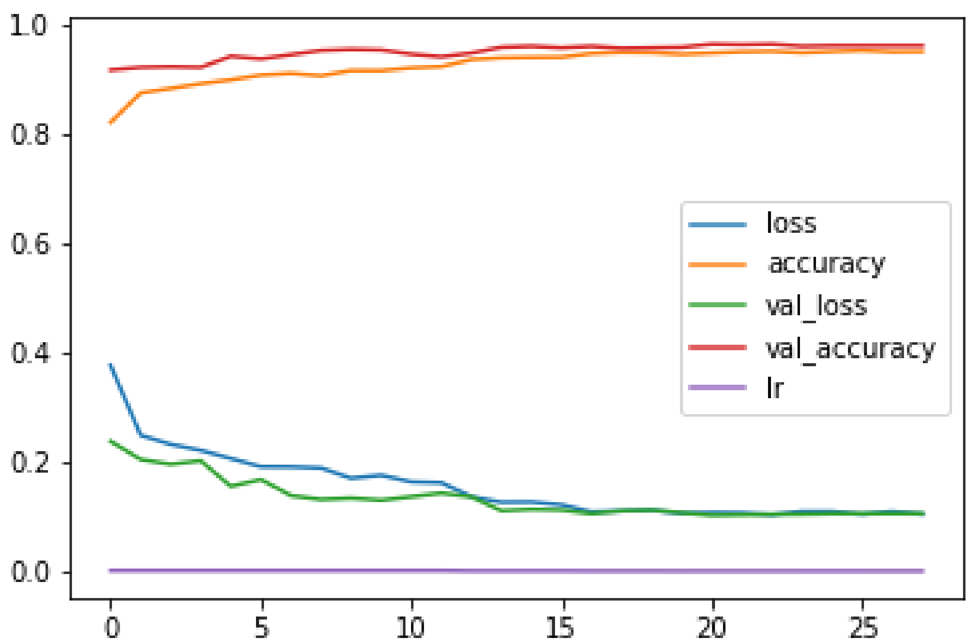
[ ]:
```
cnn = load_model('CheckPointsColored/modelMobileNet_21.pb/')
```

1.9 Evaluación del modelo

[58]:
```
cnn = model
```

[59]:
```
pd.DataFrame(cnn.history.history).plot()
```

[59]: <AxesSubplot:>



[66]:
```
test_accu = cnn.evaluate(test)
print('The testing accuracy is :',test_accu[1]*100, '%')
```

```
65/65 [==============================] - 25s 379ms/step - loss: 0.1498 -
accuracy: 0.9403
The testing accuracy is : 94.03272271156311 %
```

[61]:
```
predictions = cnn.predict(test,verbose=1)
```

```
65/65 [==============================] - 22s 324ms/step
```

[62]:
```
preds = np.argmax(predictions.copy(),1)
```

[63]:
```
from sklearn.metrics import classification_report,confusion_matrix
columns=["Predicted Covid", "Predicted Normal","Predicted other"]
classification_metrics=classification_report(test.
 ↪classes,preds,target_names=columns)
print(classification_metrics)
```

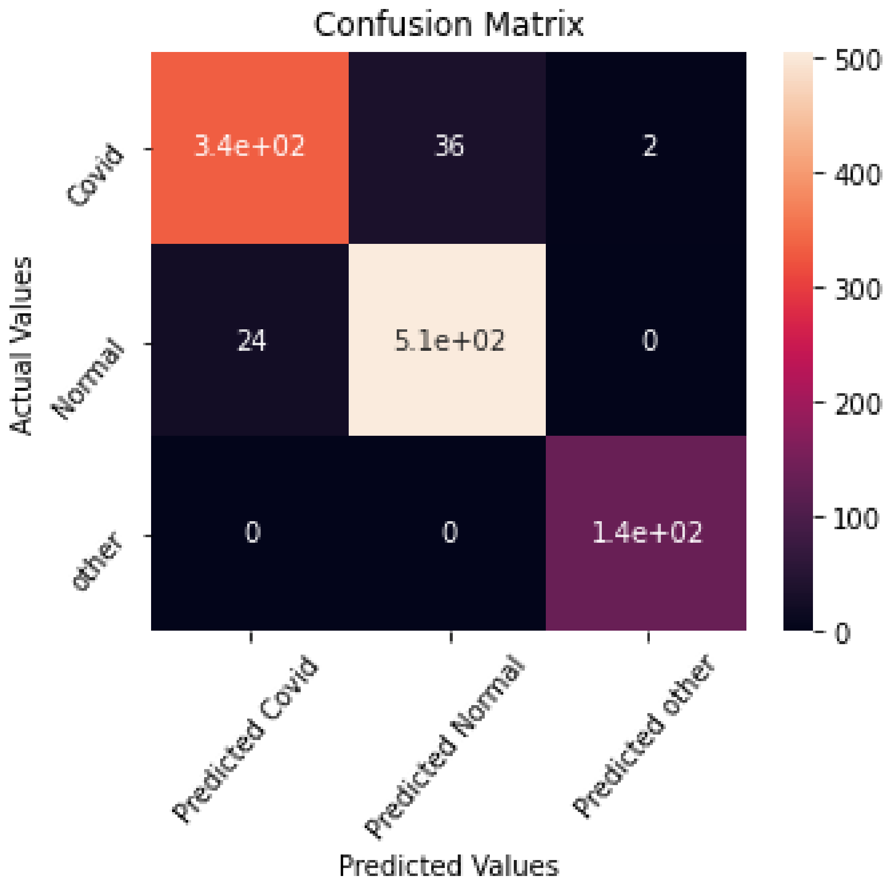
```
                  precision    recall  f1-score   support

 Predicted Covid       0.93      0.90      0.92       374
Predicted Normal       0.93      0.95      0.94       530
 Predicted other       0.99      1.00      0.99       135

        accuracy                           0.94      1039
       macro avg       0.95      0.95      0.95      1039
    weighted avg       0.94      0.94      0.94      1039
```

[64]:
```
confusion_mtx = confusion_matrix(test.classes, preds)
cm_df = pd.DataFrame(confusion_mtx,
                     index = ['Covid','Normal','other'],
                     columns = columns)
```

[65]:
```
import seaborn as sns
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actual Values')
plt.xticks(rotation=50)
plt.yticks(rotation = 50)
plt.xlabel('Predicted Values')
plt.show()
```



1.10 Testeo con imagenes propias

Podemos realizar pruebas utilizando el modelo con nuestras propias imágenes.

[82]:
```
# Testing with my own Chest X-Ray
my_path = 'ImagenesPropias/prueba.jpg'
from tensorflow.keras.preprocessing import image
my_img = image.load_img(my_path, target_size=(224, 224))
# Preprocessing the image
pp_my_img = image.img_to_array(my_img)
pp_my_img = pp_my_img/255
pp_my_img = np.expand_dims(pp_my_img, axis=0)
#predict
predictions= cnn.predict(pp_my_img)
#print
plt.figure(figsize=(20,20))
plt.axis('off')
out = ('{:.2%} COVID,  {:.2%} NORMAL,  {:.2%} VIRAL '.
 ↪format(predictions[0][0],predictions[0][1],predictions[0][2]))
```

```python
plt.title("Analisis COVID X-Ray\n"+out)

plt.imshow(np.squeeze(pp_my_img))

plt.savefig('prueba.jpg', bbox_inches='tight')
plt.show()
```

[108]:
```python
from tensorflow.keras.preprocessing import image
my_img = load_images_from_folder('ImagenesPropias')
# Preprocessing the image

for i in range(len(my_img)):
```

```
  pp_my_img = image.img_to_array(my_img[i][0])
  pp_my_img = pp_my_img/255
  pp_my_img = np.expand_dims(pp_my_img, axis=0)
  #predict
  predictions= cnn.predict(pp_my_img)
  #print
  plt.figure(figsize=(10,10))
  plt.axis('off')
  out = ('{:.2%} COVID,  {:.2%} NORMAL,  {:.2%} VIRAL '.
↪format(predictions[0][0],predictions[0][1],predictions[0][2]))
  plt.title("Analisis de "+my_img[i][1]+" COVID X-Ray: \n"+out)
  plt.imshow(np.squeeze(pp_my_img))

plt.show()
```

Analisis de COVID-30.png COVID X-Ray:
85.92% COVID, 14.01% NORMAL, 0.08% VIRAL

Analisis de COVID-41.png COVID X-Ray:
20.00% COVID, 80.00% NORMAL, 0.00% VIRAL

Analisis de COVID-74.png COVID X-Ray:
99.67% COVID, 0.33% NORMAL, 0.00% VIRAL

Analisis de COVID-75.png COVID X-Ray:
99.99% COVID, 0.01% NORMAL, 0.00% VIRAL

Analisis de COVID-79.png COVID X-Ray:
100.00% COVID, 0.00% NORMAL, 0.00% VIRAL

Analisis de prueba.jpg COVID X-Ray:
0.01% COVID, 99.21% NORMAL, 0.77% VIRAL

Analisis de Viral Pneumonia-102.png COVID X-Ray:
0.01% COVID, 99.21% NORMAL, 0.77% VIRAL

Analisis de Viral Pneumonia-145.png COVID X-Ray:
0.08% COVID, 12.67% NORMAL, 87.25% VIRAL

1.11 Explicación LIME

Como parte de los trabajos futuros, se plantea el uso de LIME para conseguir una mayor explicabilidad de las imágenes.

[109]:
```
!pip install lime
```

```
Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
Requirement already satisfied: matplotlib in e:\programs\anaconda\lib\site-
packages (from lime) (3.3.2)
Requirement already satisfied: numpy in e:\programs\anaconda\lib\site-packages
```

```
(from lime) (1.19.2)
Requirement already satisfied: scipy in e:\programs\anaconda\lib\site-packages
(from lime) (1.5.2)
Requirement already satisfied: tqdm in e:\programs\anaconda\lib\site-packages
(from lime) (4.50.2)
Requirement already satisfied: scikit-learn>=0.18 in
e:\programs\anaconda\lib\site-packages (from lime) (0.23.2)
Requirement already satisfied: scikit-image>=0.12 in
e:\programs\anaconda\lib\site-packages (from lime) (0.17.2)
Requirement already satisfied: kiwisolver>=1.0.1 in
e:\programs\anaconda\lib\site-packages (from matplotlib->lime) (1.3.0)
Requirement already satisfied: cycler>=0.10 in e:\programs\anaconda\lib\site-
packages (from matplotlib->lime) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in
e:\programs\anaconda\lib\site-packages (from matplotlib->lime) (2.8.1)
Requirement already satisfied: pillow>=6.2.0 in e:\programs\anaconda\lib\site-
packages (from matplotlib->lime) (8.0.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
e:\programs\anaconda\lib\site-packages (from matplotlib->lime) (2.4.7)
Requirement already satisfied: certifi>=2020.06.20 in
e:\programs\anaconda\lib\site-packages (from matplotlib->lime) (2020.6.20)
Requirement already satisfied: joblib>=0.11 in e:\programs\anaconda\lib\site-
packages (from scikit-learn>=0.18->lime) (0.17.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
e:\programs\anaconda\lib\site-packages (from scikit-learn>=0.18->lime) (2.1.0)
Requirement already satisfied: networkx>=2.0 in e:\programs\anaconda\lib\site-
packages (from scikit-image>=0.12->lime) (2.5)
Requirement already satisfied: imageio>=2.3.0 in e:\programs\anaconda\lib\site-
packages (from scikit-image>=0.12->lime) (2.9.0)
Requirement already satisfied: tifffile>=2019.7.26 in
e:\programs\anaconda\lib\site-packages (from scikit-image>=0.12->lime)
(2020.10.1)
Requirement already satisfied: PyWavelets>=1.1.1 in
e:\programs\anaconda\lib\site-packages (from scikit-image>=0.12->lime) (1.1.1)
Requirement already satisfied: six in e:\programs\anaconda\lib\site-packages
(from cycler>=0.10->matplotlib->lime) (1.15.0)
Requirement already satisfied: decorator>=4.3.0 in
e:\programs\anaconda\lib\site-packages (from networkx>=2.0->scikit-
image>=0.12->lime) (4.4.2)
Building wheels for collected packages: lime
  Building wheel for lime (setup.py): started
  Building wheel for lime (setup.py): finished with status 'done'
  Created wheel for lime: filename=lime-0.2.0.1-py3-none-any.whl size=283850
sha256=75811f31ff2996356f17def36de1341c149c3cedf3298c03341dd5ace20b0ff6
  Stored in directory: c:\users\nanis\appdata\local\pip\cache\wheels\e6\a6\20\cc
1e293fcdb67ede666fed293cb895395e7ecceb4467779546
Successfully built lime
Installing collected packages: lime
```

```
Successfully installed lime-0.2.0.1
```

[7]:
```python
import lime
import numpy as np
```

[5]:
```python
from lime import lime_image
explainer = lime_image.LimeImageExplainer()
```

[9]:
```python
# Testing with my own Chest X-Ray
my_path = 'ImagenesPropias/COVID.png'
from tensorflow.keras.preprocessing import image
my_img = image.load_img(my_path, target_size=(224, 224))
# Preprocessing the image
pp_my_img = image.img_to_array(my_img)
pp_my_img = pp_my_img/255
pp_my_img = np.expand_dims(pp_my_img, axis=0)
#predict
predictions= cnn.predict(pp_my_img)
```

[10]:
```python
%%time
# Hide color is the color for a superpixel turned OFF. Alternatively, if it is␣
↪NONE, the superpixel will be replaced by the average of its pixels
explanation = explainer.explain_instance(pp_my_img[0].astype('double'), cnn.
↪predict, top_labels=3, hide_color=0, num_samples=1000)
```

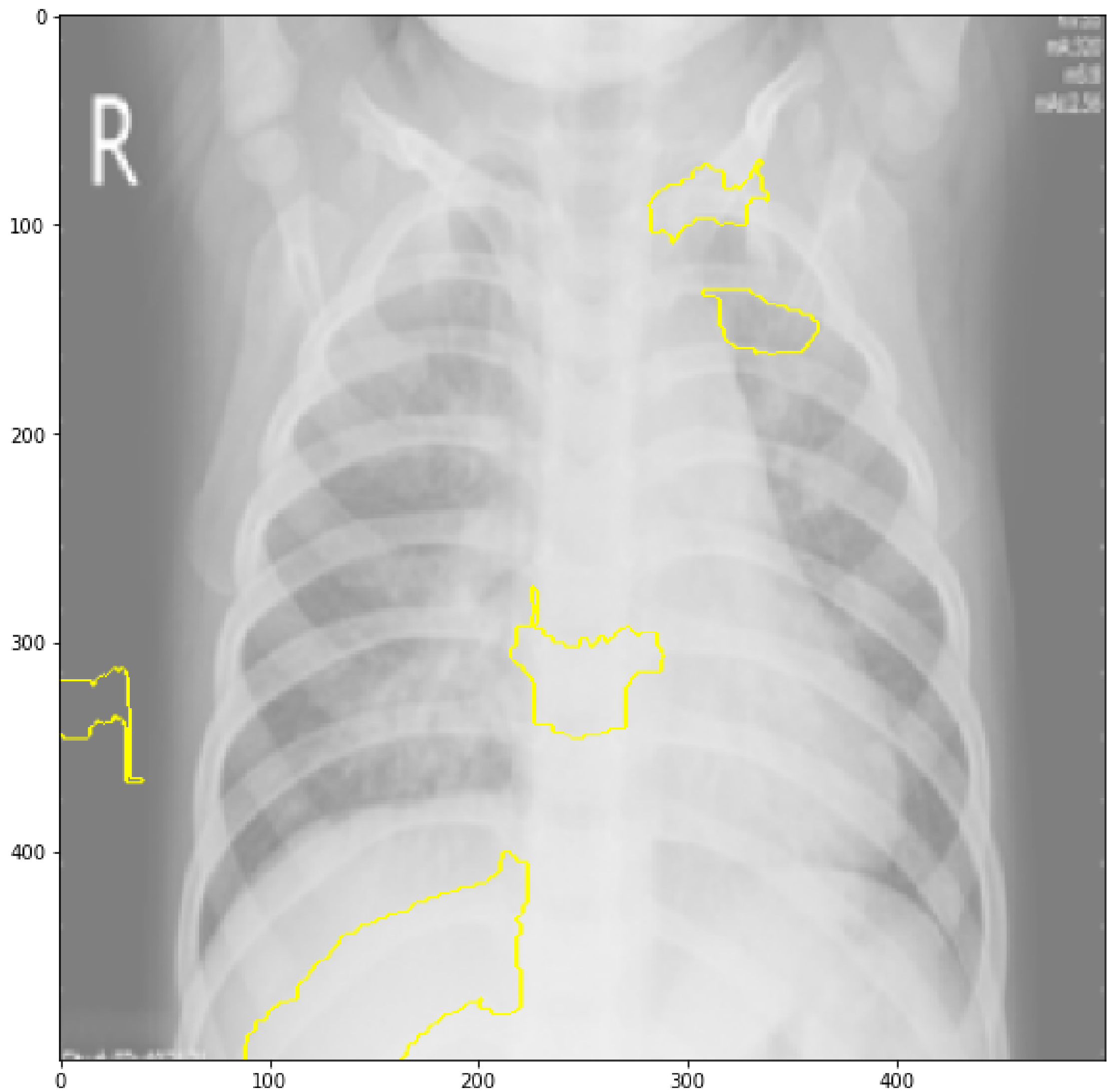
```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0),␣
↪HTML(value='')))


Wall time: 24.5 s
```

[11]:
```python
from skimage.segmentation import mark_boundaries
```

[156]:
```python
temp, mask = explanation.get_image_and_mask(explanation.top_labels[0],␣
↪positive_only=True, num_features=5, hide_rest=False)
plt.figure(figsize=(10,10))
plt.imshow(mark_boundaries(temp / 2 + 0.5, mask))
```

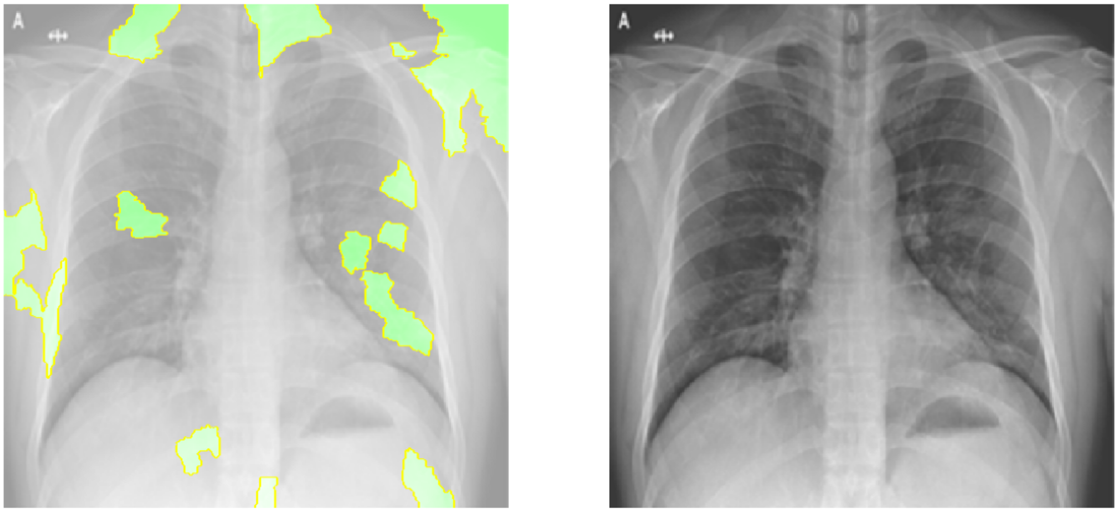
[156]:
```
<matplotlib.image.AxesImage at 0x1a483018070>
```




```
[22]: temp, mask = explanation.get_image_and_mask(explanation.top_labels[0],␣
       ↪positive_only=False, num_features=20, hide_rest=False)
      fig, (ax1,ax2) = plt.subplots(1,2,figsize=(20,20))

      # out = ('{:.2%} COVID,  {:.2%} NORMAL,  {:.2%} VIRAL '.
       ↪format(predictions[0][0],predictions[0][1],predictions[0][2]))
      # ax1.title("Analisis de "+my_img[i][1]+" COVID X-Ray: \n"+out)
      ax1.imshow(mark_boundaries(temp / 2 + 0.5, mask))
      ax2.imshow(my_img)
      ax1.axis('off')
      ax2.axis('off')
```

```
[22]: (-0.5, 499.5, 499.5, -0.5)
```

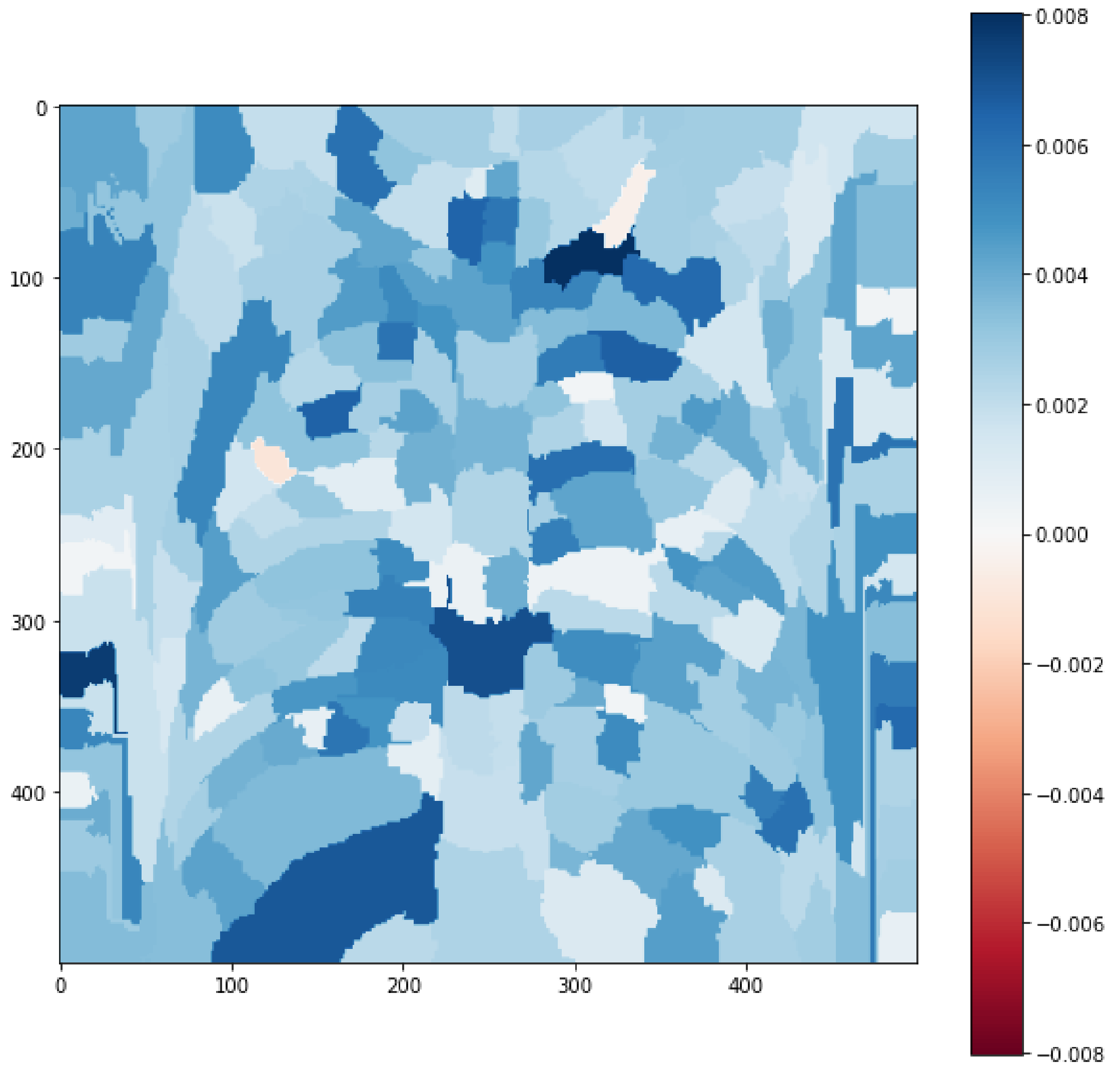


```
[158]: #Select the same class explained on the figures above.
       ind =  explanation.top_labels[0]

       #Map each explanation weight to the corresponding superpixel
       dict_heatmap = dict(explanation.local_exp[ind])
       heatmap = np.vectorize(dict_heatmap.get)(explanation.segments)

       #Plot. The visualization makes more sense if a symmetrical colorbar is used.
       plt.figure(figsize=(10,10))
       plt.imshow(heatmap, cmap = 'RdBu', vmin  = -heatmap.max(), vmax = heatmap.max())
       plt.colorbar()
```

```
[158]: <matplotlib.colorbar.Colorbar at 0x1a487938bb0>
```



1.12 GRAD CAM

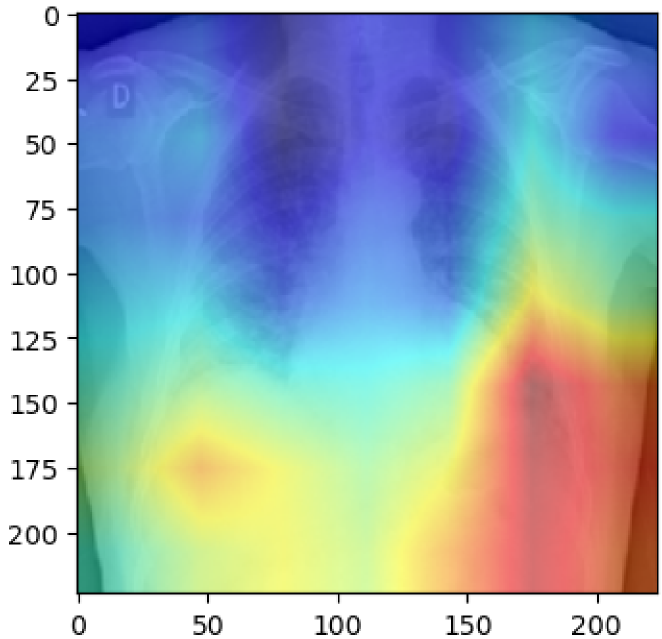
Método que nos ayuda también con la explicabilidad de las imágenes, pero mediante un mapa de calor.

[9]:
```
import gradcam
```

[22]:
```
from gradcam import VizGradCAM
from tensorflow.keras.preprocessing.image import img_to_array,load_img
```

[40]:
```
test_img = img_to_array(load_img("ImagenesPropias/COVID-6.png" ,␣
↪target_size=(224,224)))
```

[41]:
```
VizGradCAM(cnn, test_img)
```


0
25
50
75
100
125
150
175
200
0
50
100
150
200