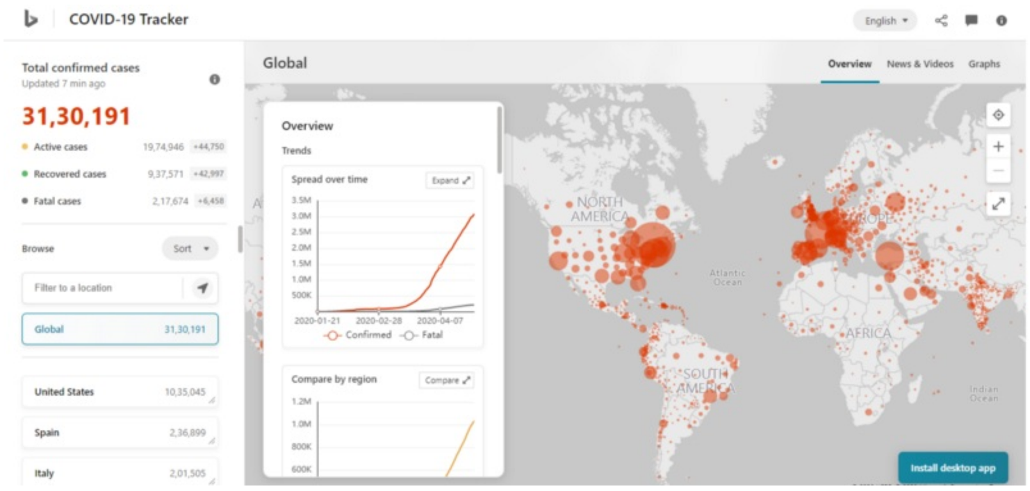
Detecting COVID-19 using Deep Learning

A practical approach to help medical practitioners helping us in the battle against COVID-19

Coronavirus disease 2019 (**COVID-19**) is a highly infectious disease caused by *severe acute respiratory syndrome coronavirus 2*. The disease first originated in December 2019 from *Wuhan, China* and since then it has spread globally across the world affecting more than **200 countries**. The impact is such that the ***World Health Organization(WHO)*** has declared the ongoing pandemic of **COVID-19** a Public Health Emergency of International Concern.

As of 29th April, there is a total of 31,30,191 cases with 2,17,674 deaths in more than 200 countries across the world. (Source: Bing COVID-19 tracker).



Total confirmed cases
Updated 7 min ago
31,30,191

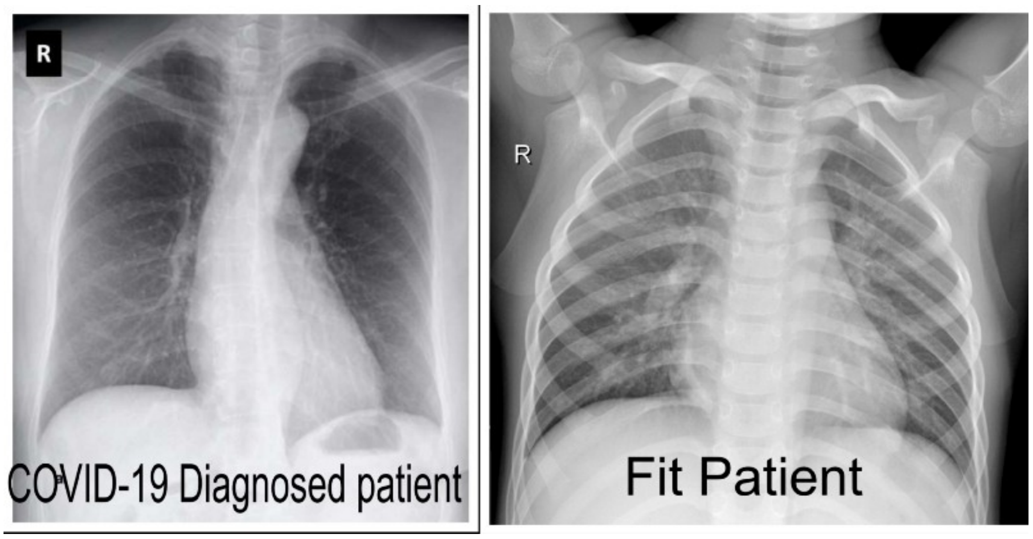
Active cases	19,74,946	+44,750
Recovered cases	9,37,571	+42,997
Fatal cases	2,17,674	+6,458

Global	31,30,191
United States	10,35,045
Spain	2,36,899
Italy	2,01,505

So, in this particular scenario, one primary thing that needs to be done and has already started in the majority of the countries is *Manual testing*, so that the true situation can be understood and appropriate decisions can be taken.

But the drawbacks of manual testing includes sparse availability of testing kits, costly and inefficient blood tests; a blood test takes around 5–6 hours to generate the result.

So, the idea is to overcome these circumstances using the Deep Learning technique for better and efficient treatment. Since the disease is highly contagious therefore as early as we generate the results the fewer cases in the city that's why we can use Convolution Neural Network to get our job done.



Can you distinguish between both X-rays if they haven't been labeled? I bet you can't, but a CNN can.

Analysis of COVID-19 using deep learning includes lungs x-rays of patients and the basic idea is to classify the x-ray as COVID affected or normal. In short, the problem is a binary classification problem where we classify Normal vs COVID-19 cases.

There are several pros and cons of using Deep Learning to tackle such kinds of situations:

1. Pros: More time saving; less expensive; easy to operate
2. Cons: Practically we need ~100% accuracy as we can't wrongly identify the patients as it might lead to further spread of disease which is highly discouraged.

But still, this model can return good accuracies and can further can be enhanced.

Data Preparation

Machine Learning needs a lot of data to train; the data we need for this type of problem is chest X-Ray for both COVID affected and fit patients.

There is no direct link to the dataset but we can make-shift to get the data and start the operation.

1. **Dr.Joseph Paul Cohen** recently open-sourced a **database** containing chest X-ray images of patients suffering from the COVID-19 disease. The dataset used is an open-source dataset which consists of COVID-19 images from publicly available research, as well as lung images with different pneumonia-causing diseases such as SARS, Streptococcus, and Pneumocystis.
2. I have also used the **Kaggle's Chest X-ray** competitions dataset to extract X-rays of healthy patients and have sampled 100 images to have a balance with the COVID-19 available images.

So, the dataset consists of COVID-19 X-ray scan images and also the angle when the scan is taken. It turns out that the most frequently used view is the Posteroanterior view and I have considered the COVID-19 PA view X-ray scans for my analysis.

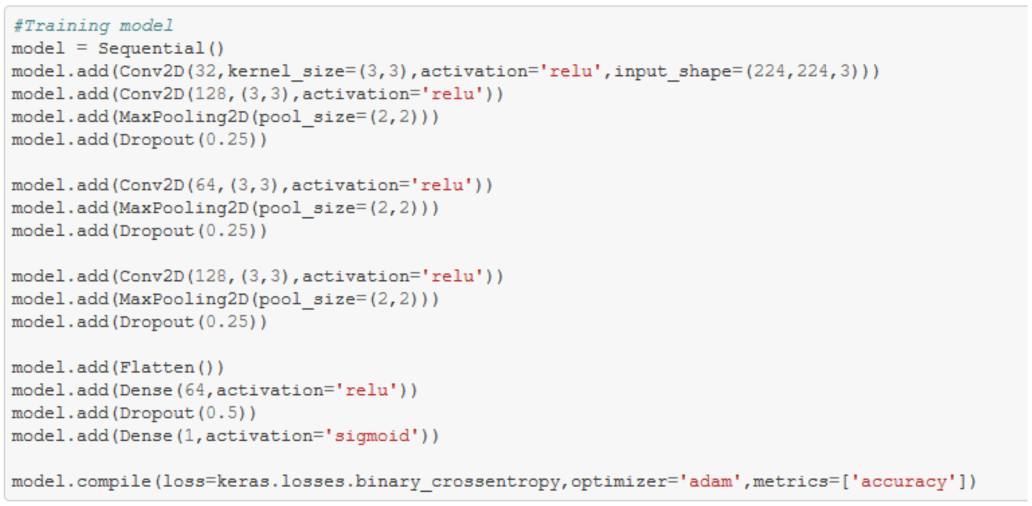
To stratify our data we will take an equal number of images and will blend them and later will divide into test and train data.

Now you all can skip the above steps if you want as I already have split and prepared the data which can be found on my ***Github Repository.***

Model Deployment

Since we have already prepared the data which is the most tedious part of this project, let's move to the next step here we will create a deep learning model that is going to learn the difference between normal X-Ray and COVID-19 affected X-Ray and later can predict.

I assume you all know the basics of CNN architecture if not I highly recommend you follow:- Basics of CNN



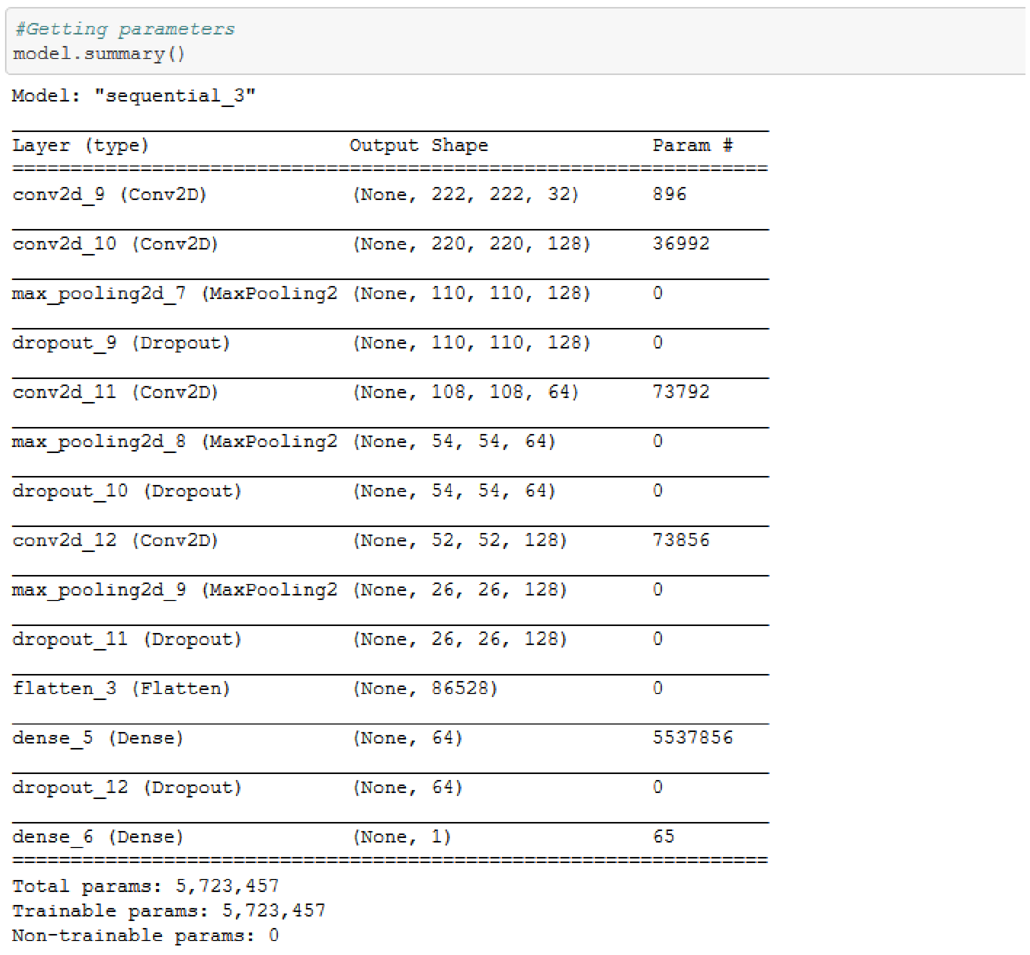
```
#Training model
model = Sequential()
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(224,224,3)))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))

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

Model Architecture

I tend to have ***3 hidden layers,*** you can experiment with more or fewer layers that is up to you. I am going to follow the traditional approach of increasing the neurons as we go deep inside the layer; as it helps to learn more features from the image which returns us better certainty.

I'm going to have *(224,224,3)* input neurons that are we are resizing our data to ***224*224 with 3 channels*** as it is considered to be the ideal size and therefore our model can grasp even minutiae and necessary features from the image.

At last, I am going to flatten our features and will use sigmoid as activation function as we are having *binary classification problem*, and thus our output will only contain one cell, ***adam as optimizer*** works pretty well with ***sigmoid*** hence compiling the model with them in addition to ***cross binary entropy.***



```
#Getting parameters
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 222, 222, 32)      896
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 220, 220, 128)     36992
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 110, 110, 128)     0
_________________________________________________________________
dropout_9 (Dropout)          (None, 110, 110, 128)     0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 108, 108, 64)      73792
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 54, 54, 64)        0
_________________________________________________________________
dropout_10 (Dropout)         (None, 54, 54, 64)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 52, 52, 128)       73856
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 26, 26, 128)       0
_________________________________________________________________
dropout_11 (Dropout)         (None, 26, 26, 128)       0
_________________________________________________________________
flatten_3 (Flatten)          (None, 86528)             0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                5537856
_________________________________________________________________
dropout_12 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 65
=================================================================
Total params: 5,723,457
Trainable params: 5,723,457
Non-trainable params: 0
```

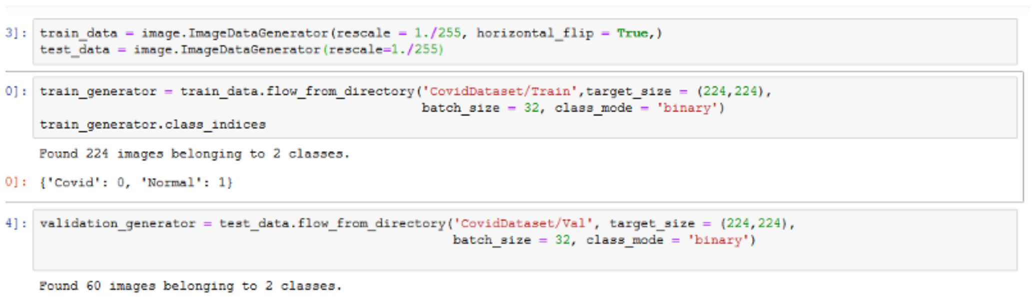
Parameters

You might be wondering why I didn't directly deploy VGG16 or any predefined model but for that, you must know the architecture of ***VGG16***, it contains roughly around ***140million parameters***, on the other hand, our model includes around ***5.7million parameters***, so it is better and more optimal to use customized model rather than training for hours on transfer learning especially for small datasets like this.

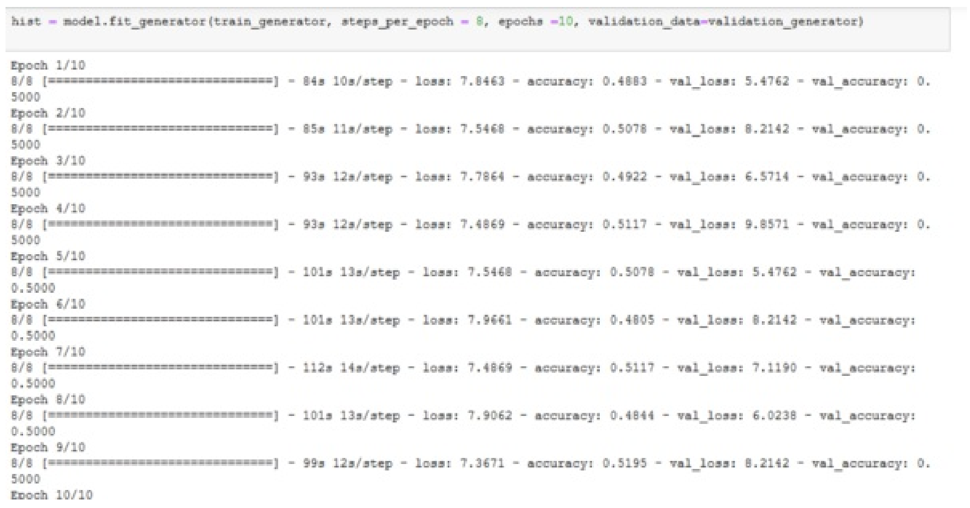
Training Data

Since we've already defined our model the next task we are left with is training our data on the defined model.

I tried training the data without performing shearing, zooming and horizontal shifts so the accuracy I got was around 50% which is pretty low for real-time projects like this.



```
3]: train_data = image.ImageDataGenerator(rescale = 1./255, horizontal_flip = True,)
    test_data = image.ImageDataGenerator(rescale=1./255)

0]: train_generator = train_data.flow_from_directory('CovidDataset/Train',target_size = (224,224),
                                                     batch_size = 32, class_mode = 'binary')
    train_generator.class_indices

    Found 224 images belonging to 2 classes.

0]: {'Covid': 0, 'Normal': 1}

4]: validation_generator = test_data.flow_from_directory('CovidDataset/Val', target_size = (224,224),
                                                         batch_size = 32, class_mode = 'binary')

    Found 60 images belonging to 2 classes.
```

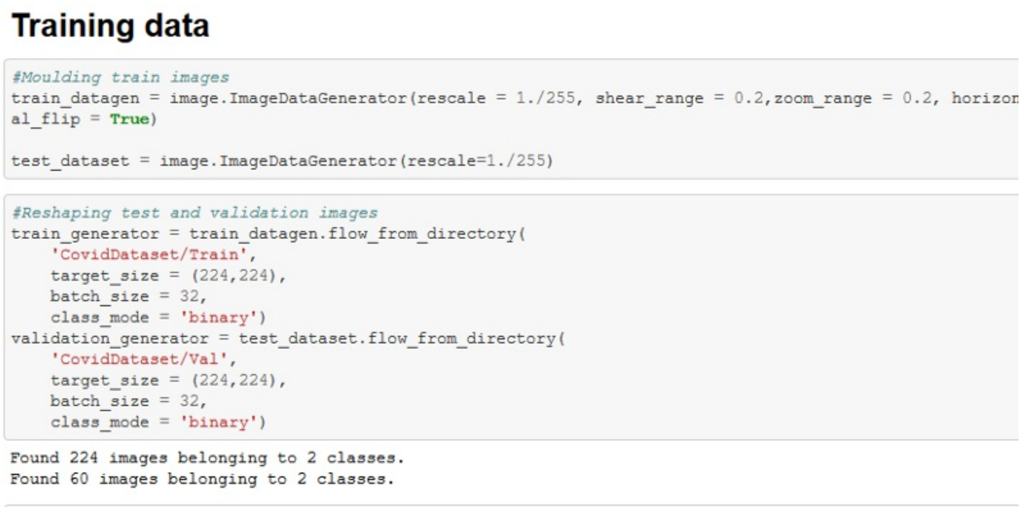


```
hist = model.fit_generator(train_generator, steps_per_epoch = 8, epochs =10, validation_data=validation_generator)

Epoch 1/10
8/8 [==============================] - 84s 10s/step - loss: 7.8463 - accuracy: 0.4883 - val_loss: 5.4762 - val_accuracy: 0.
5000
Epoch 2/10
8/8 [==============================] - 85s 11s/step - loss: 7.5468 - accuracy: 0.5078 - val_loss: 8.2142 - val_accuracy: 0.
5000
Epoch 3/10
8/8 [==============================] - 93s 12s/step - loss: 7.7864 - accuracy: 0.4922 - val_loss: 6.5714 - val_accuracy: 0.
5000
Epoch 4/10
8/8 [==============================] - 93s 12s/step - loss: 7.4869 - accuracy: 0.5117 - val_loss: 9.8571 - val_accuracy: 0.
5000
Epoch 5/10
8/8 [==============================] - 101s 13s/step - loss: 7.5468 - accuracy: 0.5078 - val_loss: 5.4762 - val_accuracy:
0.5000
Epoch 6/10
8/8 [==============================] - 101s 13s/step - loss: 7.9661 - accuracy: 0.4805 - val_loss: 8.2142 - val_accuracy:
0.5000
Epoch 7/10
8/8 [==============================] - 112s 14s/step - loss: 7.4869 - accuracy: 0.5117 - val_loss: 7.1190 - val_accuracy:
0.5000
Epoch 8/10
8/8 [==============================] - 101s 13s/step - loss: 7.9062 - accuracy: 0.4844 - val_loss: 6.0238 - val_accuracy:
0.5000
Epoch 9/10
8/8 [==============================] - 99s 12s/step - loss: 7.3671 - accuracy: 0.5195 - val_loss: 8.2142 - val_accuracy: 0.
5000
Epoch 10/10
```

Accuracy was low due to less reception as data wasn't molded

So it's better to mold the data for better reception of features, therefore, we are performing shearing, zooming, and horizontal rotation on our training data.

Once the images are sculpted we can convert the given images in the input shape that is ***224*224*** with a ***batch size of 32*** and can train our training set.



Training data

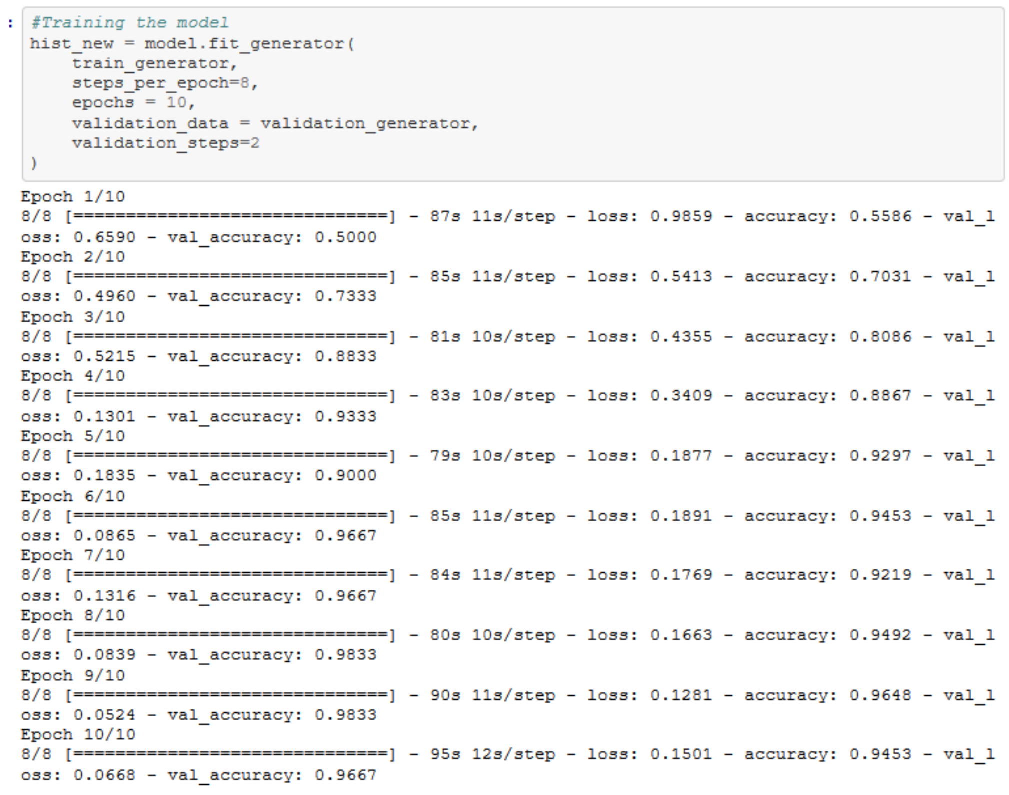
```
#Moulding train images
train_datagen = image.ImageDataGenerator(rescale = 1./255, shear_range = 0.2,zoom_range = 0.2, horizon
al_flip = True)

test_dataset = image.ImageDataGenerator(rescale=1./255)
```

```
#Reshaping test and validation images
train_generator = train_datagen.flow_from_directory(
    'CovidDataset/Train',
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'binary')
validation_generator = test_dataset.flow_from_directory(
    'CovidDataset/Val',
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'binary')
```

```
Found 224 images belonging to 2 classes.
Found 60 images belonging to 2 classes.
```

Preparing training data

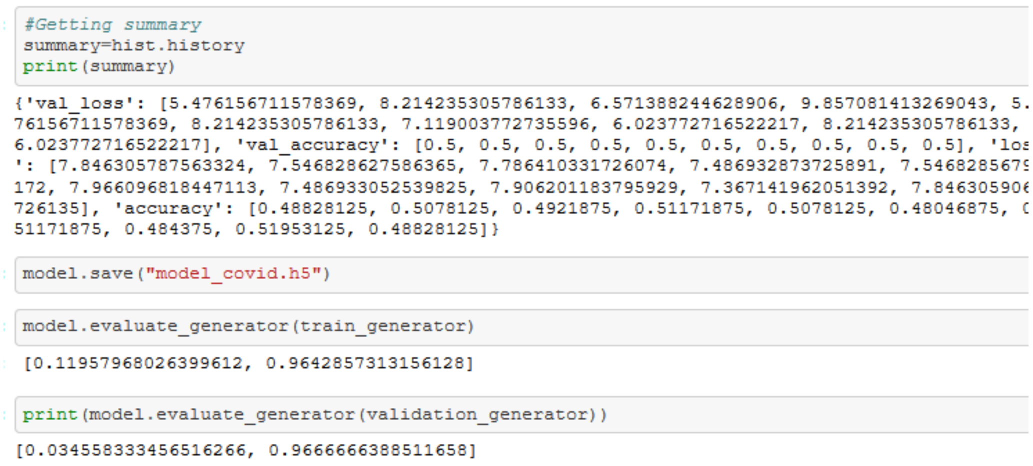
In my training process I am going with ***10 epochs with 8 steps per epochs***, again feel free to experiment with hyperparameters and maybe that could yield better results.



```
: #Training the model
hist_new = model.fit_generator(
    train_generator,
    steps_per_epoch=8,
    epochs = 10,
    validation_data = validation_generator,
    validation_steps=2
)
```

```
Epoch 1/10
8/8 [==============================] - 87s 11s/step - loss: 0.9859 - accuracy: 0.5586 - val_l
oss: 0.6590 - val_accuracy: 0.5000
Epoch 2/10
8/8 [==============================] - 85s 11s/step - loss: 0.5413 - accuracy: 0.7031 - val_l
oss: 0.4960 - val_accuracy: 0.7333
Epoch 3/10
8/8 [==============================] - 81s 10s/step - loss: 0.4355 - accuracy: 0.8086 - val_l
oss: 0.5215 - val_accuracy: 0.8833
Epoch 4/10
8/8 [==============================] - 83s 10s/step - loss: 0.3409 - accuracy: 0.8867 - val_l
oss: 0.1301 - val_accuracy: 0.9333
Epoch 5/10
8/8 [==============================] - 79s 10s/step - loss: 0.1877 - accuracy: 0.9297 - val_l
oss: 0.1835 - val_accuracy: 0.9000
Epoch 6/10
8/8 [==============================] - 85s 11s/step - loss: 0.1891 - accuracy: 0.9453 - val_l
oss: 0.0865 - val_accuracy: 0.9667
Epoch 7/10
8/8 [==============================] - 84s 11s/step - loss: 0.1769 - accuracy: 0.9219 - val_l
oss: 0.1316 - val_accuracy: 0.9667
Epoch 8/10
8/8 [==============================] - 80s 10s/step - loss: 0.1663 - accuracy: 0.9492 - val_l
oss: 0.0839 - val_accuracy: 0.9833
Epoch 9/10
8/8 [==============================] - 90s 11s/step - loss: 0.1281 - accuracy: 0.9648 - val_l
oss: 0.0524 - val_accuracy: 0.9833
Epoch 10/10
8/8 [==============================] - 95s 12s/step - loss: 0.1501 - accuracy: 0.9453 - val_l
oss: 0.0668 - val_accuracy: 0.9667
```

Training the model



```
#Getting summary
summary=hist.history
print(summary)
```

```
{'val_loss': [5.476156711578369, 8.214235305786133, 6.571388244628906, 9.857081413269043, 5.
76156711578369, 8.214235305786133, 7.119003772735596, 6.023772716522217, 8.214235305786133,
6.023772716522217], 'val_accuracy': [0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5], 'los
': [7.846305787563324, 7.546828627586365, 7.786410331726074, 7.486932873725891, 7.546828567
172, 7.966096818447113, 7.486933052539825, 7.906201183795929, 7.367141962051392, 7.84630590
726135], 'accuracy': [0.48828125, 0.5078125, 0.4921875, 0.51171875, 0.5078125, 0.48046875, 
51171875, 0.484375, 0.51953125, 0.48828125]}
```

```
model.save("model_covid.h5")
```

```
model.evaluate_generator(train_generator)
```

```
[0.11957968026399612, 0.9642857313156128]
```

```
print(model.evaluate_generator(validation_generator))
```

```
[0.034558333456516266, 0.9666666388511658]
```

Summarizing the model

Also, we can plot loss and accuracy for better understanding of required hyperparameters.

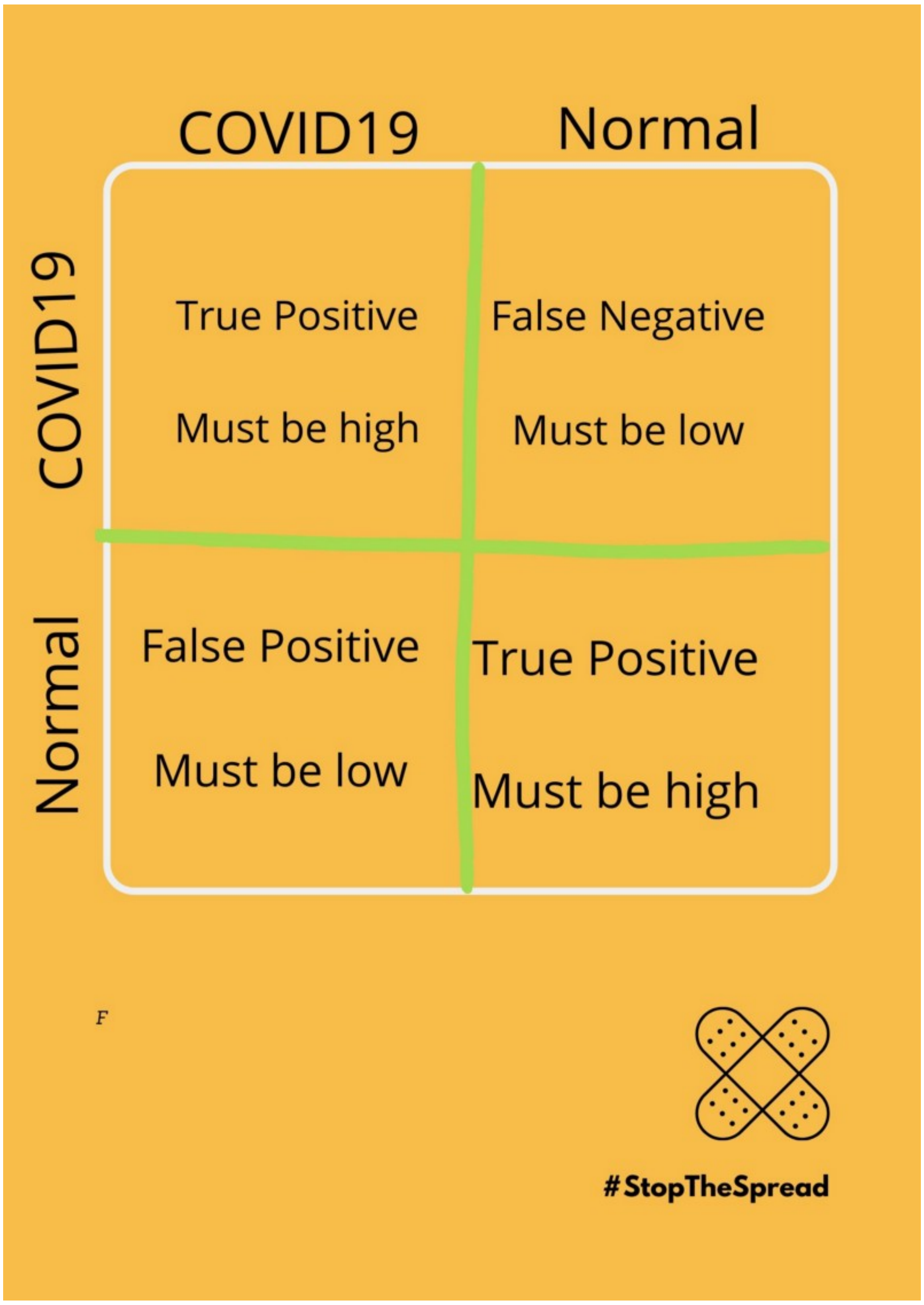
The defined hyperparameters produce 96.4% accuracy, which isn’t bad but still can be improved as, if we deploy a model with around 96.4% accuracy in real scenarios, wrongly identified patients still can spread the disease and our goal for an efficient approach wouldn’t be able to achieve success.

Again if you want to save your time from training you can download the trained model from my ***Github repository***.

Confusion Matrix

To visualize the results in a more understanding manner we’re going to implement a confusion matrix.

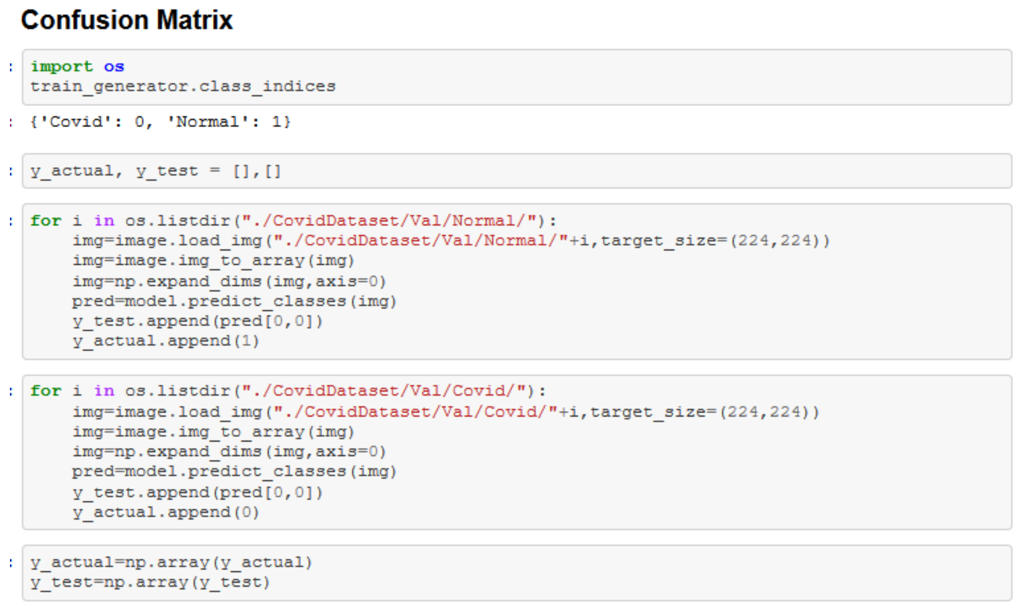
The prototype for the confusion matrix is as follows:-



	COVID19	Normal
COVID19	True Positive Must be high	False Negative Must be low
Normal	False Positive Must be low	True Positive Must be high

F
#StopTheSpread

The confusion matrix we’re getting is as follows:-



Confusion Matrix

```
import os
train_generator.class_indices
```

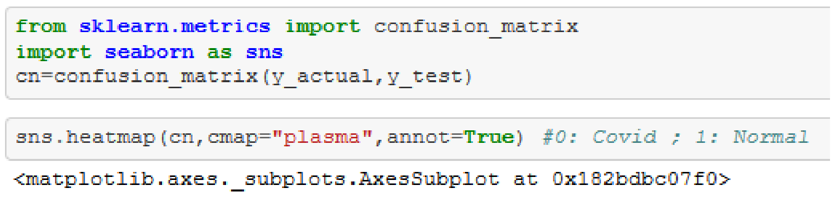
```
{'Covid': 0, 'Normal': 1}
```

```
y_actual, y_test = [],[]
```

```
for i in os.listdir("./CovidDataset/Val/Normal/"):
    img=image.load_img("./CovidDataset/Val/Normal/"+i,target_size=(224,224))
    img=image.img_to_array(img)
    img=np.expand_dims(img,axis=0)
    pred=model.predict_classes(img)
    y_test.append(pred[0,0])
    y_actual.append(1)
```

```
for i in os.listdir("./CovidDataset/Val/Covid/"):
    img=image.load_img("./CovidDataset/Val/Covid/"+i,target_size=(224,224))
    img=image.img_to_array(img)
    img=np.expand_dims(img,axis=0)
    pred=model.predict_classes(img)
    y_test.append(pred[0,0])
    y_actual.append(0)
```

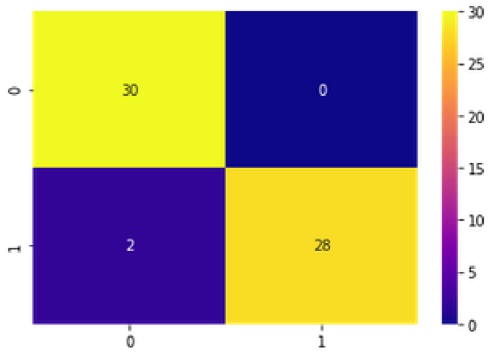
```
y_actual=np.array(y_actual)
y_test=np.array(y_test)
```



```
from sklearn.metrics import confusion_matrix
import seaborn as sns
cn=confusion_matrix(y_actual,y_test)
```

```
sns.heatmap(cn,cmap="plasma",annot=True) #0: Covid ; 1: Normal
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x182bdbc07f0>
```



Decoding the confusion matrix, out of 30 COVID affected patients we are getting *30 people* we are getting *0 wrongly classified* and, out of 28 *normal patients* we are getting *28 patients* are classified right and *2 as wrongly classified.*

The results we are getting is good but still, the accuracy can be improved in order to fulfil our intention.

Source Code

The whole source code along with dataset and trained model can be found at my Github Repository:- ***Covid-19 Detection***

Conclusion

So, to conclude, I want to ponder on the fact again that the analysis has been done on a limited dataset and the results are exploratory and nothing conclusive can be inferred from the same. Medical validations have not been done on the approach.

I plan to improve the model to increase the tenacity of my model with more X-ray scans so that the model is generalizable. Furthermore, I encourage readers to experiment on the model to make it more precise.

As always, thanks so much for reading, and please share this article if you found it useful!

Stay home, Stay safe!

Feel free to connect:

> Join me at www.dakshtrehan.com
>
> *LinkedIN ~ https://www.linkedin.com/in/dakshtrehan/*
>
> *Instagram ~ https://www.instagram.com/_daksh_trehan_/*
>
> *Github ~ https://github.com/dakshtrehan*

Read my other articles :-

> Logistic Regression Explained
>
> Linear Regression Explained
>
> Determining perfect fit for your ML Model
>
> Relating Machine Learning Techniques to Real-Life
>
> Serving Data Science to a Rookie

Follow for further Machine Learning/ Deep Learning blogs.

> *Medium ~ https://medium.com/@dakshtrehan*
>
> *Cheers.*

The cover template and confusion matrix template was made by me on www.canva.com. The x-ray images are part of open sourced dataset available on github and kaggle. The rest pictures are from my Jupyter Notebook. Please contact me if you would like permission to use them.

Also shoutout to Prateek bhaiyaa and Coding Blocks for helping me understand these models better through their data science sessions.

By Daksh Trehan on April 29, 2020.

Canonical link

Exported from Medium on July 15, 2020.