PREDICTING WEATHER A PERSON IS HEALTHY OR SUFFERING WITH COVID-19/PNEUMONIA

DATA SET:

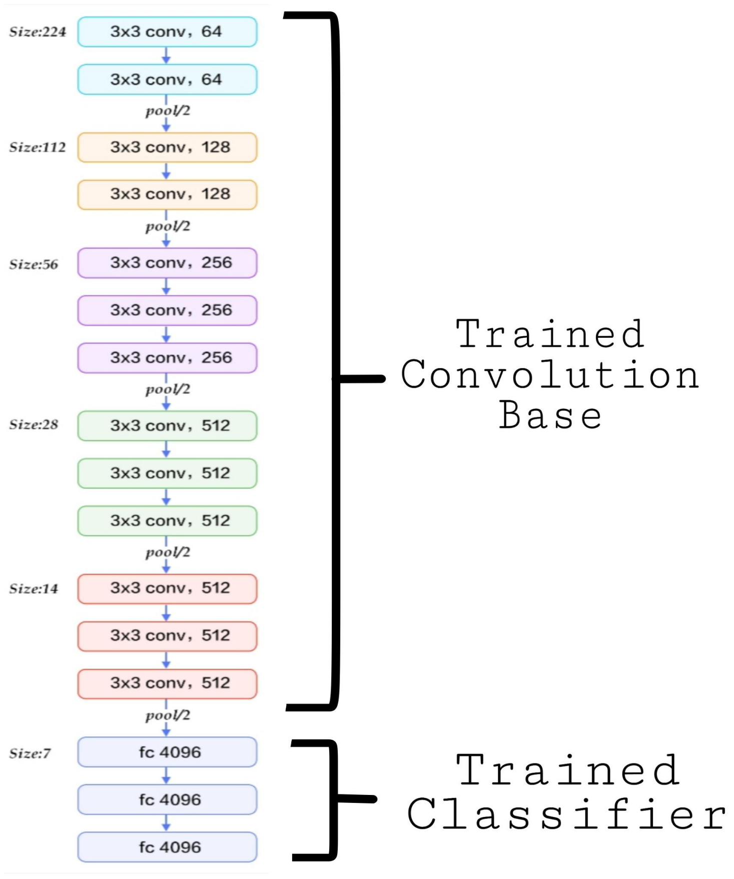
- For this project chest CT-scan images of COVID-19 patients are obtained from the open source Github repository, this repository also contains chest CT-scan images of mainly patients with ARDS, SARS, Bacterial, Chlamydophilla, Influenza, Klebsiella, Legionella, Lipoid, Pneumonia, Pneumocystis, Strptococcus, Varicella, E.colli and Mycoplasma_bacterial_pnemonia
- In addition, 50 healthy persons chest CT-scans and 50 pneumonia patients chest images were collected from Kaggle repository.

DEEP TRANFER LEARNING:

- Biggest problem faced in this problem is size of data-set is too small, to train a deep neural network to get good accuracy we need huge amount of data.
- To achieve good accuray, even with smaller data-sets, we used the concept of transfer learning.

How we trained model using transfer learning:

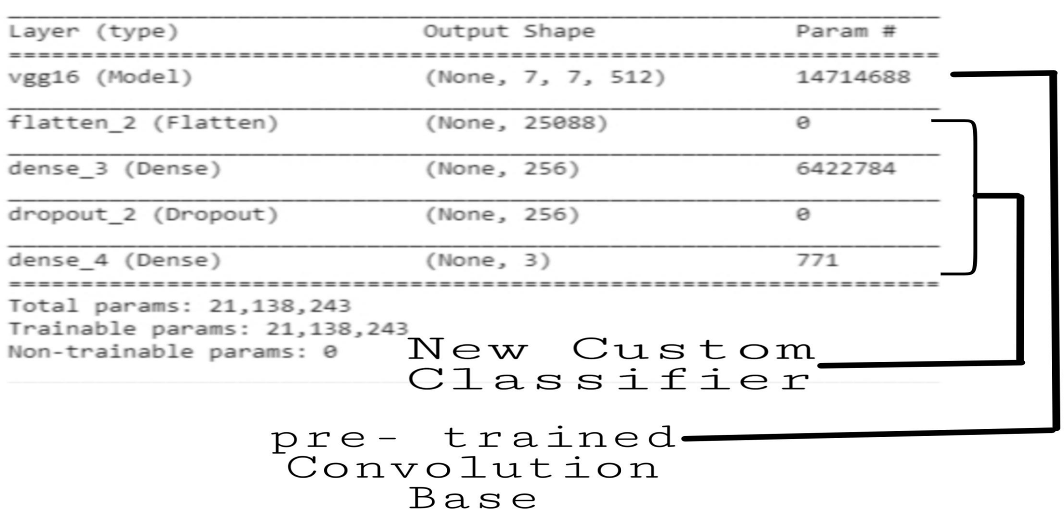
1. We take VGG-16 model which was already trained on Image-Net data-set.



2. Discarded the trained classifier and the we added our custom classifier on top of pre-trained convolution base.

   This new classifier consists of

Flatten layer : its takers 2D/3D/4D tensors and covert them into 1D tensor (ie; 1D array).
Dense layer : this fully connected layer consists of 256 neurons in it , activation function used in each neuron is ‘Relu’.
Dropout layer : as we see number of trainable parameters are too high, so our model may overfit the data. Since dropout layer is acted as regularizer, it will reduce the effect of overfitting.
Dense layer : this is the output layer which contains 3 neurons and the activation functions used in these layers is ‘Softmax’.



```
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_2 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               6422784
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 771
=================================================================
Total params: 21,138,243
Trainable params: 21,138,243
Non-trainable params: 0
```

3. Then we compile and trained the model. Before compiling and training the model, it is important to freeze the convolution base (freezing layer/set of layers means, preventing their weights from being updated during training).

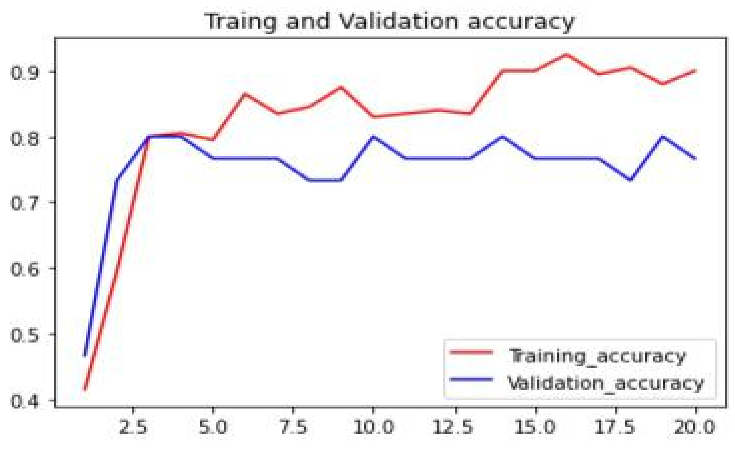
METHODOLOGY:

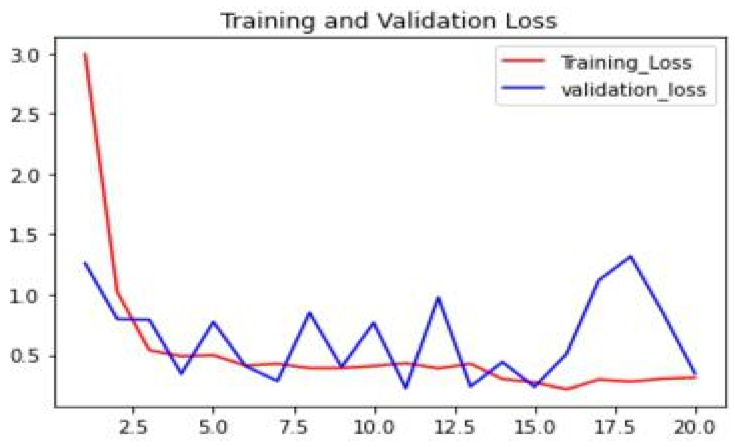
- In pre-processing phase, data augmentation and normalization is done.
- We augmented the images randomly by applying rotation, shifting, zooming, horizontal flipping, to make our model invarient to shifting, zooming, horizontal flipping and rotation.
- We applied min-max normalization on each image.The reason to apply normalization is because a small change in the input at the input layer leads to large change in the inputs at deeper layers.
- For the architecture used in CNN is VGG-16, as the results for using the architecture has given efficient results compared to other architectures.
- For the weight initialization in fully connected layers we used glorot uniform weight initialization.
- Activation function that is used in fully connected layers is Relu. Relu activation function tends to show a greater performance compared to other activation functions. Main advantage of using Relu function is there won’t be problems regarding varnishing gradient or exploding gradient and also can converge to the solution fast.

- The optimizer used in the process is Adam optimizer. The main advantages of Adam optimizer are:
  (1) It doesn't get stuck at saddle points.
  (2) It can converge to solution faster, and
  (3) For each weight it has different learning rate which means no manual tuning of learning rate is required.
- As the neural network have deeper layers, there is a good chance of overfitting. So,in order to overcome this, dropout layers can be used. The dropout layers actually acts as regularizer.

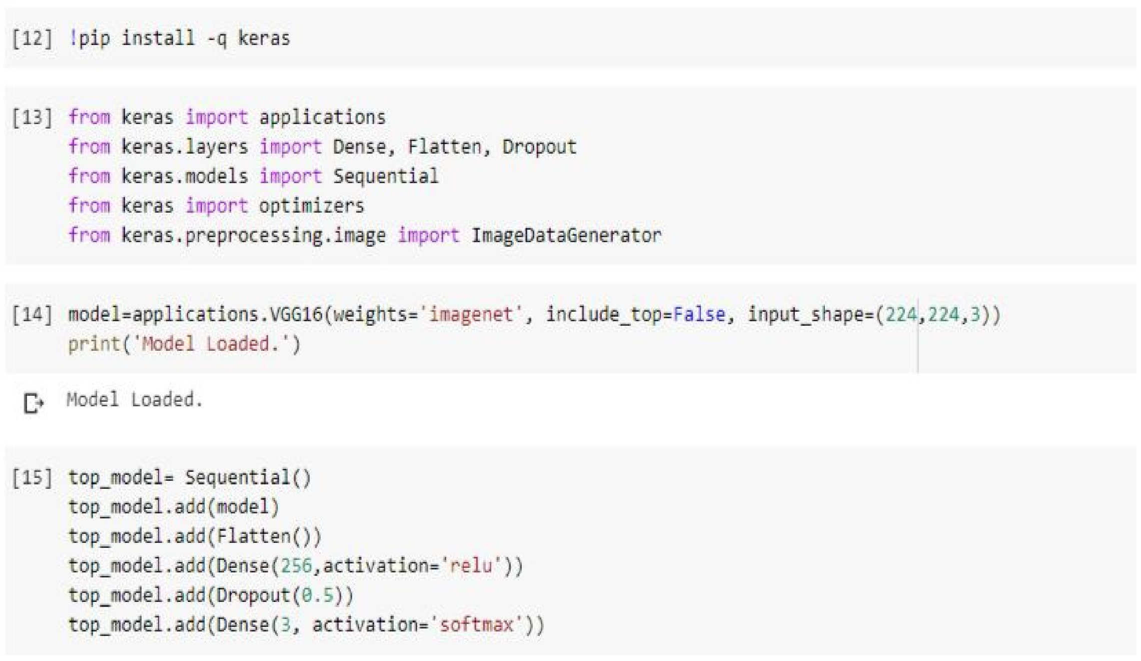
RESULT:

- A loss function is used to optimize a machine learning model. The loss is calculated on training and validation and its interpretation is based on how well the model is doing . Loss function implies how poorly or well a model behaves after each iteration.
- The loss function we have used in our model is **categorical cross entropy loss.**
- An accuracy metric is used to measure the algorithm's performance in an interpretable way. The accuracy of a model is usually determined after the model parameters and is calculated in the form of percentage. It is the measure of how accurate our model's prediction is compared to the true data.
- The overall accuracy, we get is 90%





CODE :



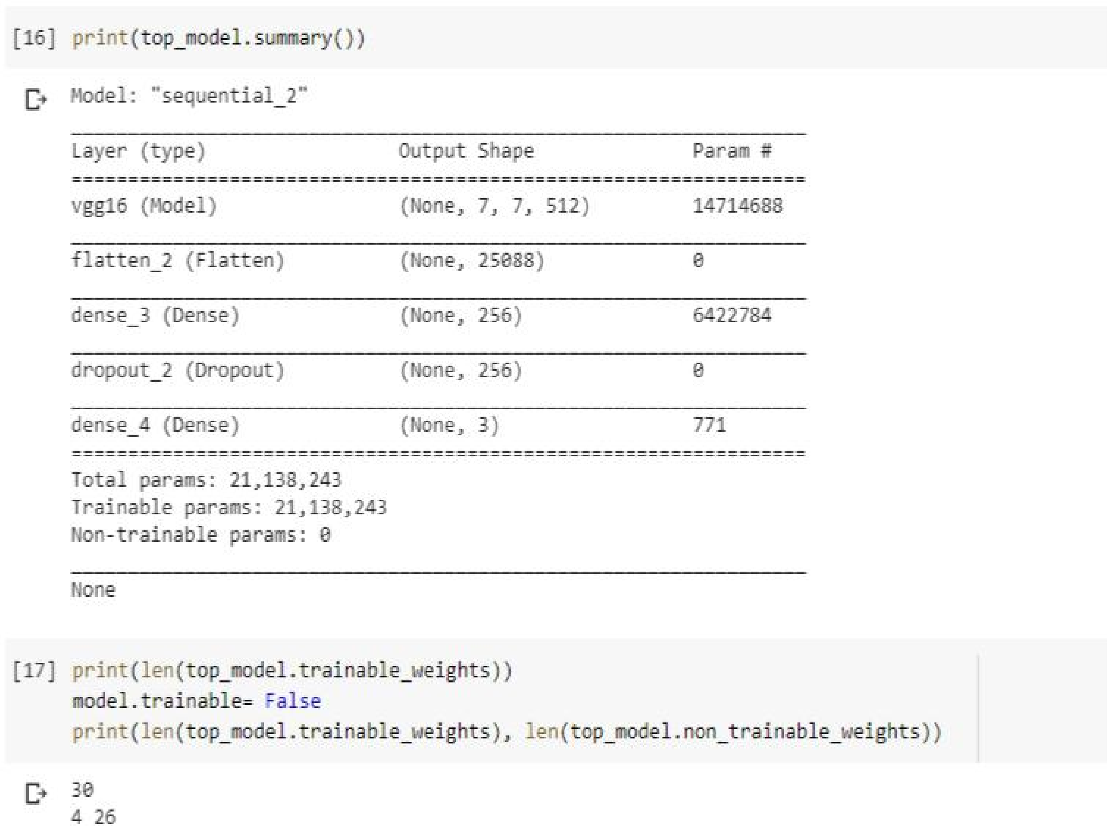
```
[12] !pip install -q keras
```

```
[13] from keras import applications
     from keras.layers import Dense, Flatten, Dropout
     from keras.models import Sequential
     from keras import optimizers
     from keras.preprocessing.image import ImageDataGenerator
```

```
[14] model=applications.VGG16(weights='imagenet', include_top=False, input_shape=(224,224,3))
     print('Model Loaded.')
```

```
Model Loaded.
```

```
[15] top_model= Sequential()
     top_model.add(model)
     top_model.add(Flatten())
     top_model.add(Dense(256,activation='relu'))
     top_model.add(Dropout(0.5))
     top_model.add(Dense(3, activation='softmax'))
```

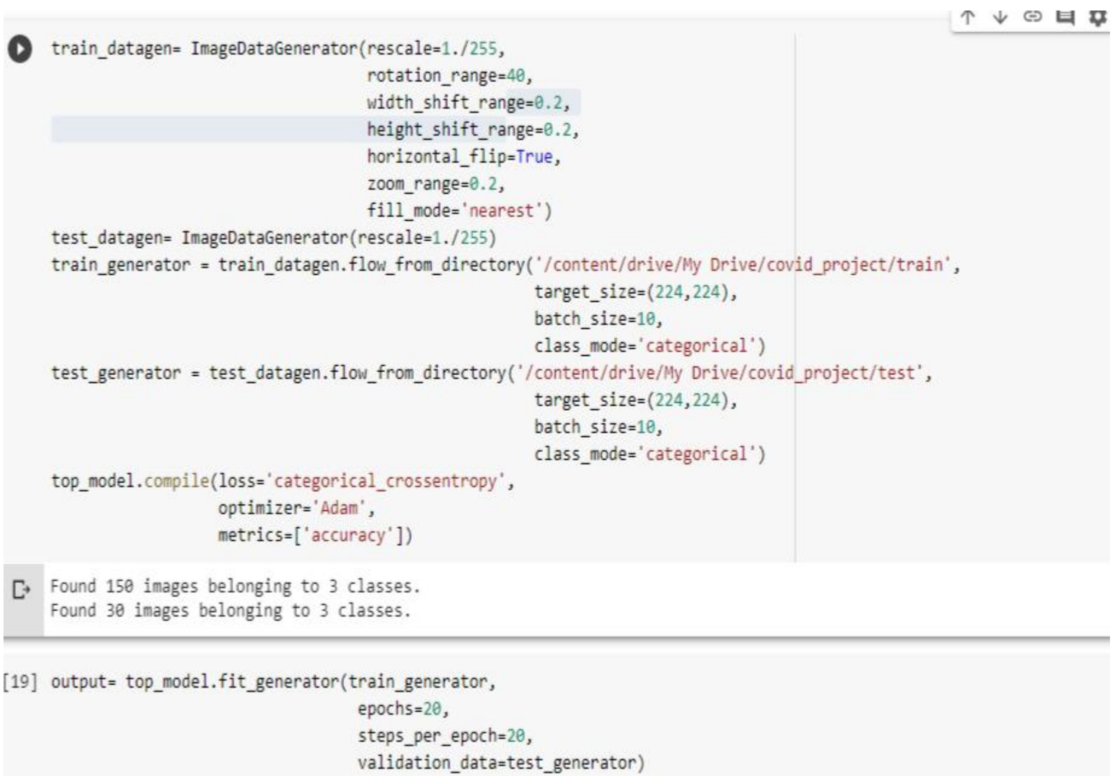


```
[16] print(top_model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_2 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               6422784
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 771
=================================================================
Total params: 21,138,243
Trainable params: 21,138,243
Non-trainable params: 0
_________________________________________________________________
None
```

```
[17] print(len(top_model.trainable_weights))
     model.trainable= False
     print(len(top_model.trainable_weights), len(top_model.non_trainable_weights))
```

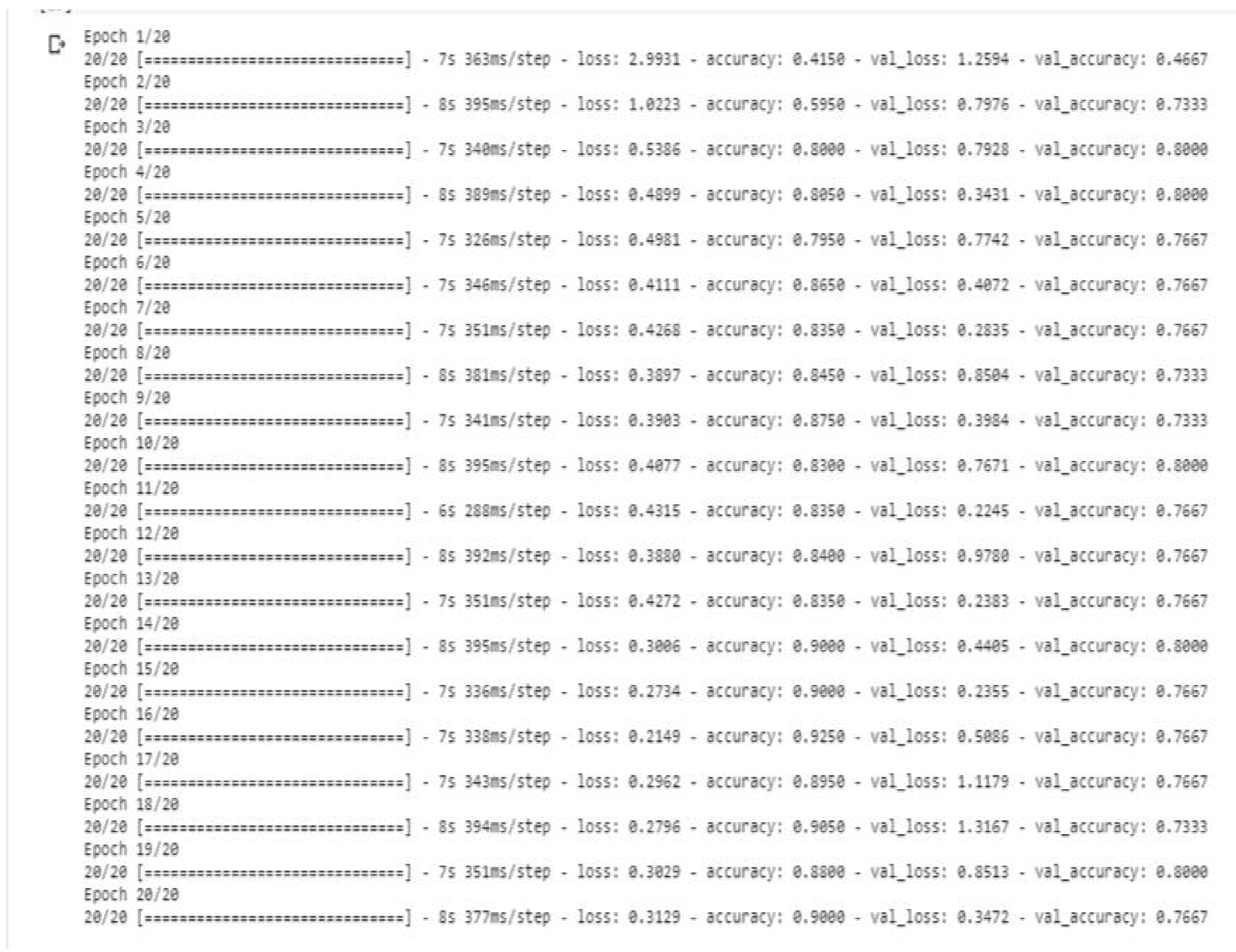
```
30
4 26
```



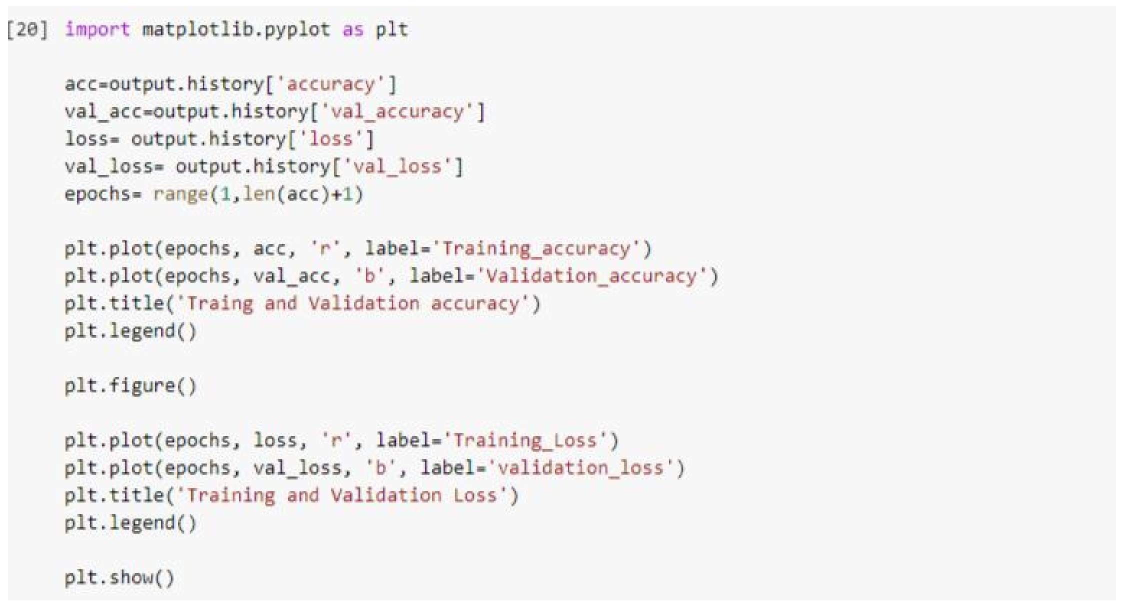
```
train_datagen= ImageDataGenerator(rescale=1./255,
                                  rotation_range=40,
                                  width_shift_range=0.2,
                                  height_shift_range=0.2,
                                  horizontal_flip=True,
                                  zoom_range=0.2,
                                  fill_mode='nearest')
test_datagen= ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory('/content/drive/My Drive/covid_project/train',
                                                    target_size=(224,224),
                                                    batch_size=10,
                                                    class_mode='categorical')
test_generator = test_datagen.flow_from_directory('/content/drive/My Drive/covid_project/test',
                                                    target_size=(224,224),
                                                    batch_size=10,
                                                    class_mode='categorical')
top_model.compile(loss='categorical_crossentropy',
                  optimizer='Adam',
                  metrics=['accuracy'])
```

```
Found 150 images belonging to 3 classes.
Found 30 images belonging to 3 classes.
```

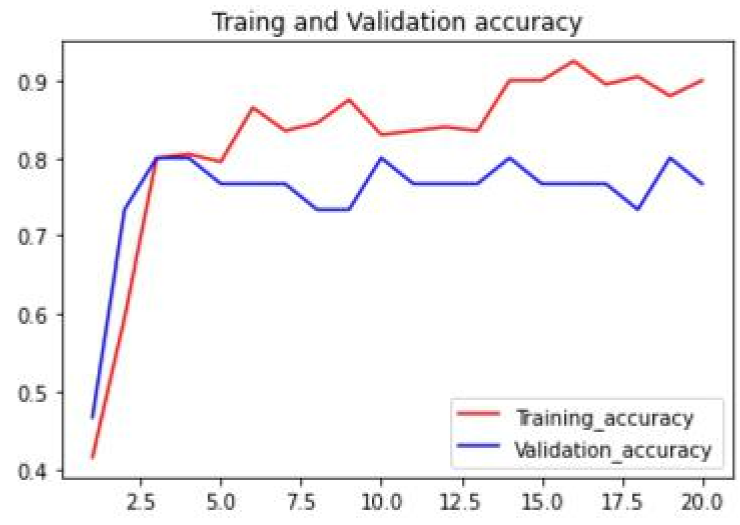
```
[19] output= top_model.fit_generator(train_generator,
                                  epochs=20,
                                  steps_per_epoch=20,
                                  validation_data=test_generator)
```

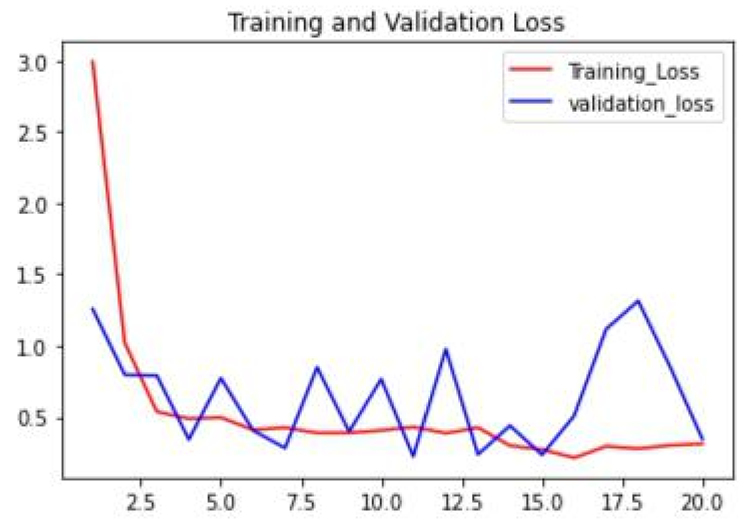


```
Epoch 1/20
20/20 [==============================] - 7s 363ms/step - loss: 2.9931 - accuracy: 0.4150 - val_loss: 1.2594 - val_accuracy: 0.4667
Epoch 2/20
20/20 [==============================] - 8s 395ms/step - loss: 1.0223 - accuracy: 0.5950 - val_loss: 0.7976 - val_accuracy: 0.7333
Epoch 3/20
20/20 [==============================] - 7s 340ms/step - loss: 0.5386 - accuracy: 0.8000 - val_loss: 0.7928 - val_accuracy: 0.8000
Epoch 4/20
20/20 [==============================] - 8s 389ms/step - loss: 0.4899 - accuracy: 0.8050 - val_loss: 0.3431 - val_accuracy: 0.8000
Epoch 5/20
20/20 [==============================] - 7s 326ms/step - loss: 0.4981 - accuracy: 0.7950 - val_loss: 0.7742 - val_accuracy: 0.7667
Epoch 6/20
20/20 [==============================] - 7s 346ms/step - loss: 0.4111 - accuracy: 0.8650 - val_loss: 0.4072 - val_accuracy: 0.7667
Epoch 7/20
20/20 [==============================] - 7s 351ms/step - loss: 0.4268 - accuracy: 0.8350 - val_loss: 0.2835 - val_accuracy: 0.7667
Epoch 8/20
20/20 [==============================] - 8s 381ms/step - loss: 0.3897 - accuracy: 0.8450 - val_loss: 0.8504 - val_accuracy: 0.7333
Epoch 9/20
20/20 [==============================] - 7s 341ms/step - loss: 0.3903 - accuracy: 0.8750 - val_loss: 0.3984 - val_accuracy: 0.7333
Epoch 10/20
20/20 [==============================] - 8s 395ms/step - loss: 0.4077 - accuracy: 0.8300 - val_loss: 0.7671 - val_accuracy: 0.8000
Epoch 11/20
20/20 [==============================] - 6s 288ms/step - loss: 0.4315 - accuracy: 0.8350 - val_loss: 0.2245 - val_accuracy: 0.7667
Epoch 12/20
20/20 [==============================] - 8s 392ms/step - loss: 0.3880 - accuracy: 0.8400 - val_loss: 0.9780 - val_accuracy: 0.7667
Epoch 13/20
20/20 [==============================] - 7s 351ms/step - loss: 0.4272 - accuracy: 0.8350 - val_loss: 0.2383 - val_accuracy: 0.7667
Epoch 14/20
20/20 [==============================] - 8s 395ms/step - loss: 0.3006 - accuracy: 0.9000 - val_loss: 0.4405 - val_accuracy: 0.8000
Epoch 15/20
20/20 [==============================] - 7s 336ms/step - loss: 0.2734 - accuracy: 0.9000 - val_loss: 0.2355 - val_accuracy: 0.7667
Epoch 16/20
20/20 [==============================] - 7s 338ms/step - loss: 0.2149 - accuracy: 0.9250 - val_loss: 0.5086 - val_accuracy: 0.7667
Epoch 17/20
20/20 [==============================] - 7s 343ms/step - loss: 0.2962 - accuracy: 0.8950 - val_loss: 1.1179 - val_accuracy: 0.7667
Epoch 18/20
20/20 [==============================] - 8s 394ms/step - loss: 0.2796 - accuracy: 0.9050 - val_loss: 1.3167 - val_accuracy: 0.7333
Epoch 19/20
20/20 [==============================] - 7s 351ms/step - loss: 0.3029 - accuracy: 0.8800 - val_loss: 0.8513 - val_accuracy: 0.8000
Epoch 20/20
20/20 [==============================] - 8s 377ms/step - loss: 0.3129 - accuracy: 0.9000 - val_loss: 0.3472 - val_accuracy: 0.7667
```



```
[20] import matplotlib.pyplot as plt

     acc=output.history['accuracy']
     val_acc=output.history['val_accuracy']
     loss= output.history['loss']
     val_loss= output.history['val_loss']
     epochs= range(1,len(acc)+1)

     plt.plot(epochs, acc, 'r', label='Training_accuracy')
     plt.plot(epochs, val_acc, 'b', label='Validation_accuracy')
     plt.title('Traing and Validation accuracy')
     plt.legend()

     plt.figure()

     plt.plot(epochs, loss, 'r', label='Training_Loss')
     plt.plot(epochs, val_loss, 'b', label='validation_loss')
     plt.title('Training and Validation Loss')
     plt.legend()

     plt.show()
```


Traing and Validation accuracy
Training_accuracy
Validation_accuracy


Training and Validation Loss
Training_Loss
validation_loss