Detection of COVID-19 from Chest X-rays through CNN implemented from Python Keras API

Corona Virus the most spoken pandemic in recent days whose origin still remains a mystery, is a highly infectious disease caused by severe acute respiratory syndrome 2. Almost most of the countries in the world are dumbstruck by this epidemic. Almost every physician around the world are working on its prevention, cure and detection optimisation techniques. In an attempt to help the detection of this disease I devised a convolution neural networks using Python Keras API with the dataset of Kaggle's Chest Xray Pneumonia detection competition.

Overview of Dataset:

From Kaggle's Chest X_ray dataset, I extracted 100 images of normal Chest X_ray i.e healthy person without COVID19 disease and another set of 100 images of COVID19 chest X-ray images. With the help of this I started to devise Convolution Neural Networks by following procedure:

With the help of Python Keras API I imported the necessary functions required for our network build. I imported optimizers such as ADAM, SGD, RMSPROP in an attempt to reduce the losses by making them tune the parameters such as weights and learning rate. I imported vgg16 to build convolution neural network model.

```
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator,load_img, img_to_array
from keras.models import Sequential, Model
from keras.layers import Conv2D, MaxPooling2D,GlobalAveragePooling2D
from keras.layers import Activation, Dropout, BatchNormalization, Flatten, Dense

from keras.models import Sequential, Model
#from keras.applications.xception import Xception
#from keras.applications.resnet50 import ResNet50
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.optimizers import Adam, SGD, RMSprop

import tensorflow as tf

import os
import numpy as np
import pandas as np

import matplotlib.pyplot as plt
%matplotlib inline
```

```
Using TensorFlow backend.
```

With the help of python functions I imported the folder from my desktop. I designed the input shape of the model through image height, width and channel parameters and also declared other essential variables like epochs, batch size, etc.

```
In [4]: DATASET_DIR = r"C:\Users\Adarsh\Downloads\covid-19-x-ray-10000-images\dataset"

        IMG_W = 150
        IMG_H = 150
        CHANNELS = 3

        INPUT_SHAPE = (IMG_W, IMG_H, CHANNELS)
        NB_CLASSES = 2
        EPOCHS = 15
        BATCH_SIZE = 8

In [5]: os.listdir(DATASET_DIR)

Out[5]: ['covid', 'normal']
```

Here comes the definition of the Convolution Neural Network model. The model was initialised as Sequential. 3 layers of Convolution was created as its evident from the code. Filters were set to 32 for the first two and for the final layer the Convolution 2D model filter was set to 64 and kernel_size were set to (2,2). Layers *early* in the network architecture (i.e., closer to the actual input image) learn *fewer* convolutional filters while layers *deeper* in the network (i.e., closer to the output predictions) will learn *more* filters. So the model was designed by considering the above constraint. The activation function was set to 'relu' for all the three layers. In order to reduce spatial dimensions in the output the max pooling function was set to (2,2). The models attribute such as Dense support the specification of their input shape via the argument . At the end of the network layer we add a fully connected layer. Here 64 nodes are appended to CNN. Finally a classifier sigmoid activation function is added – the output of the layer are prediction values themselves.

```
In [6]: model = Sequential()
        model.add(Conv2D(32, (3, 3), input_shape=INPUT_SHAPE))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Conv2D(32, (3, 3)))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Conv2D(64, (3, 3)))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Flatten())
        model.add(Dense(64))
        model.add(Activation('relu'))
        model.add(Dropout(0.5))
        model.add(Dense(1))
        model.add(Activation('sigmoid'))
```

The model is then compiled by the following step where optimizer selected was RMSPROp which gave me less error value and loss function that I Chose was binary_cross_entropy.

```
In [7]: model.compile(loss='binary_crossentropy',
                     optimizer='rmsprop',
                     metrics=['accuracy'])
```

The model's summary is printed as follows. Using model.summary() function.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 148, 148, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 72, 72, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 34, 34, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                1183808
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_5 (Activation)    (None, 1)                 0
=================================================================
Total params: 1,212,513
Trainable params: 1,212,513
Non-trainable params: 0
```

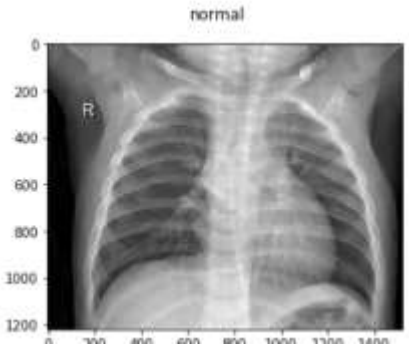
In order to check the plots of the image first image from each of the category is printed.

```
In [9]: import glob
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg
        %matplotlib inline

        normal_images = []
        for img_path in glob.glob(DATASET_DIR + '/normal/*'):
            normal_images.append(mpimg.imread(img_path))

        fig = plt.figure()
        fig.suptitle('normal')
        plt.imshow(normal_images[0], cmap='gray')

Out[9]: <matplotlib.figure.AxesImage at 0x1fd68b7c188>
```

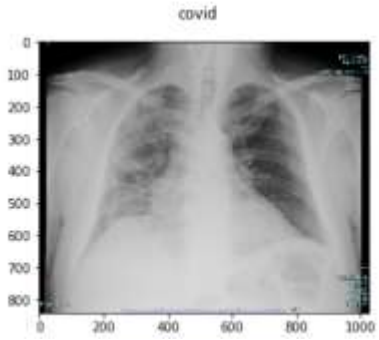


The image from covid Xray patient category is printed to have a look on it.

```
In [10]: covid_images = []
         for img_path in glob.glob(DATASET_DIR + '/covid/*'):
             covid_images.append(mpimg.imread(img_path))

         fig = plt.figure()
         fig.suptitle('covid')
         plt.imshow(covid_images[0], cmap='gray')

Out[10]: <matplotlib.figure.AxesImage at 0x1fd68c41b08>
```



Now the whole set is divided to training dataset and validation set by specifying parameters like shear_range and validation_split. Now the model is trained by using fit_generator method by specifying number of epochs, batch size of validation data and train data.

In [11]:
```
train_datagen = ImageDataGenerator(rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.4)

train_generator = train_datagen.flow_from_directory(
    DATASET_DIR,
    target_size=(IMG_H, IMG_W),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='training')

validation_generator = train_datagen.flow_from_directory(
    DATASET_DIR, # same directory as training data
    target_size=(IMG_H, IMG_W),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='validation')

history = model.fit_generator(
    train_generator,
    steps_per_epoch = train_generator.samples // BATCH_SIZE,
    validation_data = validation_generator,
    validation_steps = validation_generator.samples // BATCH_SIZE,
    epochs = EPOCHS)
```

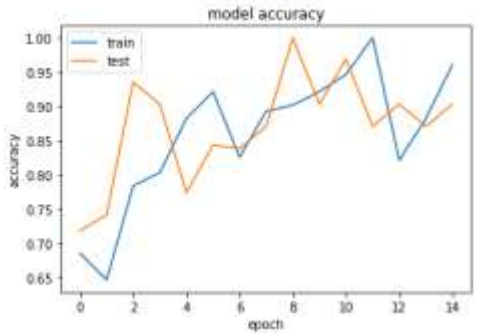
The results of precision, error and accuracy for each epoch of training is printed.

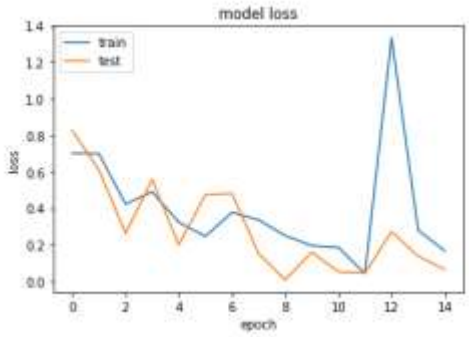
```
Epoch 1/15
7/7 [==============================] - 7s 1s/step - loss: 0.6737 - accuracy: 0.6863 - val_loss: 0.8253 - va
l_accuracy: 0.7188
Epoch 2/15
7/7 [==============================] - 6s 824ms/step - loss: 0.6908 - accuracy: 0.6471 - val_loss: 0.6098 -
val_accuracy: 0.7419
Epoch 3/15
7/7 [==============================] - 4s 557ms/step - loss: 0.4381 - accuracy: 0.7843 - val_loss: 0.2628 -
val_accuracy: 0.9355
Epoch 4/15
7/7 [==============================] - 3s 410ms/step - loss: 0.4909 - accuracy: 0.8036 - val_loss: 0.5579 -
val_accuracy: 0.9032
Epoch 5/15
7/7 [==============================] - 3s 428ms/step - loss: 0.3225 - accuracy: 0.8824 - val_loss: 0.1998 -
val_accuracy: 0.7742
Epoch 6/15
7/7 [==============================] - 3s 491ms/step - loss: 0.2354 - accuracy: 0.9216 - val_loss: 0.4736 -
val_accuracy: 0.8438
Epoch 7/15
7/7 [==============================] - 4s 604ms/step - loss: 0.3436 - accuracy: 0.8261 - val_loss: 0.4815 -
val_accuracy: 0.8387
Epoch 8/15
7/7 [==============================] - 3s 415ms/step - loss: 0.3361 - accuracy: 0.8929 - val_loss: 0.1524 -
val_accuracy: 0.8710
Epoch 9/15
7/7 [==============================] - 3s 362ms/step - loss: 0.2314 - accuracy: 0.9020 - val_loss: 0.0081 -
val_accuracy: 1.0000
Epoch 10/15
7/7 [==============================] - 3s 455ms/step - loss: 0.1813 - accuracy: 0.9216 - val_loss: 0.1594 -
val_accuracy: 0.9032
Epoch 11/15
7/7 [==============================] - 4s 528ms/step - loss: 0.1862 - accuracy: 0.9464 - val_loss: 0.0520 -
val_accuracy: 0.9688
Epoch 12/15
7/7 [==============================] - 4s 549ms/step - loss: 0.0362 - accuracy: 1.0000 - val_loss: 0.0463 -
val_accuracy: 0.8710
Epoch 13/15
7/7 [==============================] - 3s 392ms/step - loss: 1.3349 - accuracy: 0.8214 - val_loss: 0.2721 -
val_accuracy: 0.9032
Epoch 14/15
7/7 [==============================] - 3s 464ms/step - loss: 0.3033 - accuracy: 0.8824 - val_loss: 0.1380 -
val_accuracy: 0.8710
Epoch 15/15
7/7 [==============================] - 3s 414ms/step - loss: 0.1537 - accuracy: 0.9608 - val_loss: 0.0649 -
val_accuracy: 0.9032
```

Now the accuracy and loss are plotted in separate graphs. Its seen that the accuracy graph increases as the number of epochs increases and model loss decreases as the number of epochs increases.

```
In [12]: plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(['train', 'test'], loc='upper left')
         plt.show()

         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('model loss')
         plt.ylabel('loss')
         plt.xlabel('epoch')
         plt.legend(['train', 'test'], loc='upper left')
         plt.show()
```





Conclusion:

The analysis is made of less amount of dataset. Clinical trials/medical validations have not been done on the approach. If those were included with high scalability and more dataset , Covid can be detected easily with the prevailing X-ray technology which is highly reliable and need not rely for new technologies.

References :

1. https://github.com/ieee8023/covid-chestxray-dataset

2. A brilliant blog on similar lines by *Ayrton San Joaquin:* https://towardsdatascience.com/using-deep-learning-to-detect-ncov-19-from-x-ray-images-1a89701d1acd