Image Classification of Chest X-Rays using Deep Learning to predict COVID-19 disease

Rajkumar Dhanapal, Ehtesham Akhlaq Malik, Kushboo Shah, Kowshik Kumar Bandameedipalli

[1] Harrisburg University of Science and Technology

Author Note

Capstone Project for Machine Learning II course at Harrisburg University of Science Technology, 326 Market St, Harrisburg, PA 17101

Image Classification of Chest X-Rays using Deep Learning to predict COVID-19 disease

Contents

Introduction, motivation and general description of the situation, problem or challenge

Pneumonia is lung inflammation caused by infection of viruses, bacteria, and other microorganisms. It is an acute disease which can lead to multiple problems especially during old age. It is important to diagnose pneumonia early, in order to prevent as well as receive proper treatment. According to the World Health Organization, pneumonia killed more than 808,000 children under the age of 5 in 2017, accounting for 15% of all deaths of children under the age of 5 years. Detection of pneumonia has been crucial during the COVID'19 pandemic. The pandemic is still evolving with the Delta variant research is still in progress and the modality is being developed.

Diagnosis of the pneumonia condition is performed by chest X-rays, sputum, and blood tests, based on symptoms and physical examination. Pneumonia conditions can be prevented if they are identified in the initial stages but there is a certain margin of errors by radiologists while making predictions to identify the level of severity for pneumonia. Margin of error can be minimized by a technique using a deep learning algorithm especially convolutional neural networks while performing chest X-ray diagnosis. This paper will explain the implementation procedure of deep learning algorithms to detect Pneumonia conditions which will help to reduce the mortality rates.

We aim to implement a neural networks algorithm for image classification which is reliable as well as not expensive. Our focus is to create a model which will have a high accuracy compared to the existing models. The research project methodology uses deep learning algorithms which will be able to identify the level of pneumonia correctly and will help to check if the pre-processing of the chest X-ray images have retained better metrics.

Related Work:

Data Scientists and Engineers have been working pre-COVID to develop machine learning models that can help the doctors and medical staff to interpret the X rays and thereby devise effective treatment plans. During COVID, time is of crux as a result, the earlier the virus is identified and remedy taken, lesser are the chances of hospitalization and deaths. Moreover, in the widespread pandemic, X rays are a cheaper and readily available measure that can help detect COVID.

Through chest X-rays doctors can detect whether Pneumonia has occurred or not. As per prior research in the field, Pneumonia is one of the leading causes of deaths in the world. This Pneumonia can be induced through various sources, therefore identifying the exact cause of Pneumonia while looking at X-rays is of essence. Moreover, before doctors can start the treatment for COVID, they need to know the severity of the virus and damage to lungs as it has been observed that giving antibiotics at an early stage decreases the body's resistance to fight COVID and leads to complications.

The alternative remedies suggested for detecting COVID like PCR test are time consuming, have higher costs and have a higher error rate. Therefore, relying on these tests for COVID detection is not optimal or reliable. Furthermore, there is a margin of error when radiologists or technicians read the X-ray therefore the advanced machine learning models can help bridge this gap.

Our goal is to develop a higher accuracy model that is cheaper, reliable and time efficient. Some of the challenges that we encountered during the process are listed below: • The Kaggle data source is a secondary source which is extremely high in volume. To manage the size for initial analysis we only used two categories: COVID and Normal • Some of the images have noise in the form of markings. We were forced to discard a few images which had a lot of markings as it impacted the accuracy of our model • At each layer the image

quality would diminish and therefore, we applied advanced image processing techniques to improve quality after each layer. • Initially some of the images were not clear therefore, we employed image processing tools to enhance the image features thereby increasing the performance and accuracy of our models.

Some of the work performed by authors that help guide our techniques and motivation is discussed below: 1) In 2019, Aledhari and team worked on using some of the machine learning tools to classify Pneumonia images. 2) In 2017, Antin and team had confirmed that the RT-PCR technique showed a lower positive rate for Pneumonia especially in the early stage of detection. Currently, PCR test is the most prevalent methodologies being employed everywhere and as researchers had already failed to show the shortcomings of PCR therefore, we were motivated at developing this alternate, more effective and economical method. 3) In 2019, Chen and team showed how through deep learning models they can dissect an image and gather the information to determine results just like radiologists and clinicians do. 4) In 2018, Han and Davis used the two stream faster R-NN network to separate the features and cancel the noise thereby increasing the performance. 5) In 2019, prior to the outbreak of COVID, Kadam and team had explained that pneumonia was the leading cause of death globally and that it can be detected by analyzing X-rays. 6) In 2020, Zaki, Khan and Ali discussed the advantages of using advanced machine learning processes in the diagnostics of COVID. They further discussed the prior research in this context. 7) In 2020, Kushwala and Chaudhary explained that lung infection is caused by bacteria which leads to difficulty in breathing. They explained if the infection is not handled at early stages, then it might lead to hospitalization and death. 8) O'Mahony, Campbell, Carvalho, et al in 2020, explore how deep learning has helped develop the scope of image processing along with computer vision techniques like panoramic and 3-D views. The authors try to draw out similarities and differences in the processes which act as guiding principles for people working in the realm. 9) In 2017, Rajupurkar and team had used similar techniques to detect Pneumonia caused by 14 diseases by analyzing X-Rays via deep learning methods. The above-mentioned

research along with some articles formed the basis of our analysis and motivated us to improve on the process.

Data

We have used the public sourced data available on Kaggle. The data has been compartmentalized into classes Normal and COVID. The datasets are currently used for analysis by experts in terms of Pneumonia detection and accordingly we used these to train, test and validate our model.

For analytics we use Convolutional Neural network (CNN). Since the process required a large processing memory therefore, using Anaconda iPython notebook on our PC was not an option. In order to maximize our resources, we used Google Colab virtual machine for modeling.

In total we had 10,194 images in Normal class and 3616 images in COVID class as shown in Fig. 1. The dataset classes were unbalanced, but we decided to use it as deep learning models can cope with class imbalance. During the initial analysis of the images we identified that some of the images had markings while a lot of images were not clear and therefore would not have added any information to the feature engineering process. We discarded the images that had a lot of marking as they would have created a lot of white noise in our models thereby reducing the accuracy and outputs while for unclear images, we used feature engineering techniques to make the images more prominent.

Fig. 2 shows a schematic of the data pipeline we used for this study. Using Google Colab as a base tool, we imported data from the Kaggle server directly to Google Drive. Then we did processing pulling one image at a time and saving the processed image back on Google Drive. During training, the data generator directly pulled images from Google Drive and applied augmentation before feeding it to the model.

Technical Approach:

We acquired the dataset from Kaggle. Upon data exploration, we found some poor quality images possibly due to poor lighting and photography. We used sk-image preprocessing package for contrast normalization on the images by a method called Local Histogram Equalization . This improved the poor quality images while not compromising good quality images. We inspected several image normalization methods during this study which are shown applied on a poor image and a good image in Fig.3

In order to feed image data to model training, we utilized the Image Data Generator package from Keras. This package also helped in image augmentation which increases the data quantity by creating non-identical copies. The schematic representation of the augmentation methods we used are shown in Fig.4.

One of the convolutional neural network models used in this study is shown as a schematic in Fig.5. It has 4 sets for convolutional layer blocks with 2 identical layers each. We had a MaxPooling layer after each set of convolutional layers. We also had a Batch Normalization layer after a set of convolution layers. We connected the end of the convolutional network to a fully connected layer with 2 layers out of which one is an output layer. The input for the model were images sized approximately 300 pixel square with just one channel (gray). The model summary of this CNN is also shown in Fig.6. We used Keras packages to build these layers and the model. We didn't use any pre-trained model and trained the model from scratch.

We studied the performance of a VGG19 model as a reference to building and evaluating our model. We used the pre-trained VGG19 model from the Keras package and held the model parameters untrainable. We connected it to fully connected output layers and partially trained the model, i.e., only the parameters belonging to fully connected layers. The model reached upto 89% accuracy. In Fig. 9 and Fig. 10 we show the model architecture of

the reference model and the VGG19 pre-trained model working in its background. Our goal was to train our model to get a performance comparable to the reference model.

We used checkpoints and Early stopping to end the model training if there is no meaningful improvement in the evaluation metrics. We also saved model states to Google Drive to be able to retrieve later for further training.

From this project, we were able to study and successfully implement a deep learning algorithm with a Convolutional Neural Network model from scratch which can recognize and distinguish COVID Pneumonia and Normal classes from Chest X-ray or radiography images. The methodology that was adopted for implementation to retain good performance metrics and fitting the model right is the following:

1. Building the Convolutional Neural Networks model
2. Fitting the model
3. Hyperparameters Tuning
4. Performance metrics

Firstly, the objective of our project is to utilize Chest X-ray images to build a model capable of classifying the right classes of both COVID Pneumonia and Normal classes by utilizing Keras and Tensorflow libraries with Google Colab as the developmental platform for model compilation. For this we have chosen Convolutional Neural Network architecture which has ability to create a differentiation of the Chest X-ray images by aspects wherein CNN assigns importance to various aspects or objects for these images. CNN is also helpful in flattening these images and reading them pixel by pixel to consider the features for learning patterns. The Convolutional Neural Network model was designed using the Sequential model type which allows us to construct the model layer by layer. In the input layer, the 2-Dimensional Convolution filter is used as the input layer and an architecture of three interlinked layered techniques with 2-Dimensional convolution filters are implemented.

Alongside, Maxpool layer was used for image flattening and reduction of dimensionality and Dense layer was used to reduce the outputs necessary for the linkage of these interlinked layered for output layer. In the output layer, the model is developed and set the parameters to number of classes as this is a classification problem alongside Softmax activation function is used for Multi-Classification functionality of the outputs.

Then the model is fitted using the model fit generator for train and validation datasets for training the model and changing the 'epochs' parameters for better model fit and the accuracy metric obtained from the feedback of the learning curve. Further, the parameters and hyper-parameters of the Convolutional Neural Network model were fine tuned to maximize the accuracy metrics and feedback from the learning curve of accuracy vs loss functions.

In our model, the compilation of the above model implemented takes three parameters i.e., Metrics, Loss and Optimizer. The optimizer is an important parameter which has control over the learning rates wherein we used 'Adam' as the optimizer. Further we have chosen '0.001' as our learning rate which is changed every time as we retain feedback from the learning curve. Further, categorical cross-entropy as the loss function as our problem's solution was Multiclassification. For our results interpretation, we have used the metric 'accuracy' which is optimal over other metrics to resolve the model fit issues.

Our approach towards implementation of the deep learning model with Convolutional Neural Network algorithm from scratch to recognize and distinguish COVID Pneumonia and Normal classes from Chest X-ray images was accomplished with good performance metrics having accuracy about 90% and fitted the model right. The Convolutional Neural Network model implemented was able to classify the right classes of COVID Pneumonia and Normal classes for the majority of the test runs.

However, we were not able to resolve all the issues during the development stages of

the Convolutional Neural Network model to recognize and distinguish COVID Pneumonia and Normal classes from Chest X-ray images. However, in near future we would like to improve the pre-processing of the Chest X-ray images having many discrepancies. For instance, we will implement pixelation correction techniques instead of eliminating the entire image which would reduce the size of the dataset causing train-test generalization issues. Alongside, we would like this project to go through rigorous scrutiny from expert radiologists as this project is relating to medical importance. Further, we want to adopt better performing pre-trained deep learning models and bi-directional architecture models which would ramp up the performance metrics and fit the model perfectly. We would also want to try out a few combinations of model architecture like the combination of CNN with LSTM, CNN with RNN and RNN with LSTM, etc.

This work can be implemented by anyone with knowledge and experience in developing deep learning models. However, the researcher should be affine towards medical research especially with lung related diseases alongside this project needs much scrutiny from highly qualified radiologists whether the implementation procedure is carried out correctly while pre-processing the Chest X-ray images.

References

COVID-19 chest xray. (2020, May 15). Kaggle. https://www.kaggle.com/bachrr/covid-chest-xray

Chest X-Ray Images (Pneumonia). (2018, March 24). Kaggle. https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

Goldstein, E. (2020, October 3). COVID-19 Classification of X-ray Images Using Deep Neural Networks. ArXiv.Org. https://arxiv.org/abs/2010.01362

Rajpurkar, P. (2017, November 14). CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays. . . ArXiv.Org. https://arxiv.org/abs/1711.05225

Al-Waisy, A. S., Al-Fahdawi, S., Mohammed, M. A., Abdulkareem, K. H., Mostafa, S. A., Maashi, M. S., Arif, M., & Garcia-Zapirain, B. (2020). COVID-CheXNet: hybrid deep learning framework for identifying COVID-19 virus in chest X-rays images. Soft Computing. Published. https://doi.org/10.1007/s00500-020-05424-3

Habib, N., Hasan, M. M., Reza, M. M., & Rahman, M. M. (2020). Ensemble of CheXNet and VGG-19 Feature Extractor with Random Forest Classifier for Pediatric Pneumonia Detection. SN Computer Science, 1(6). https://doi.org/10.1007/s42979-020-00373-y

Annexure

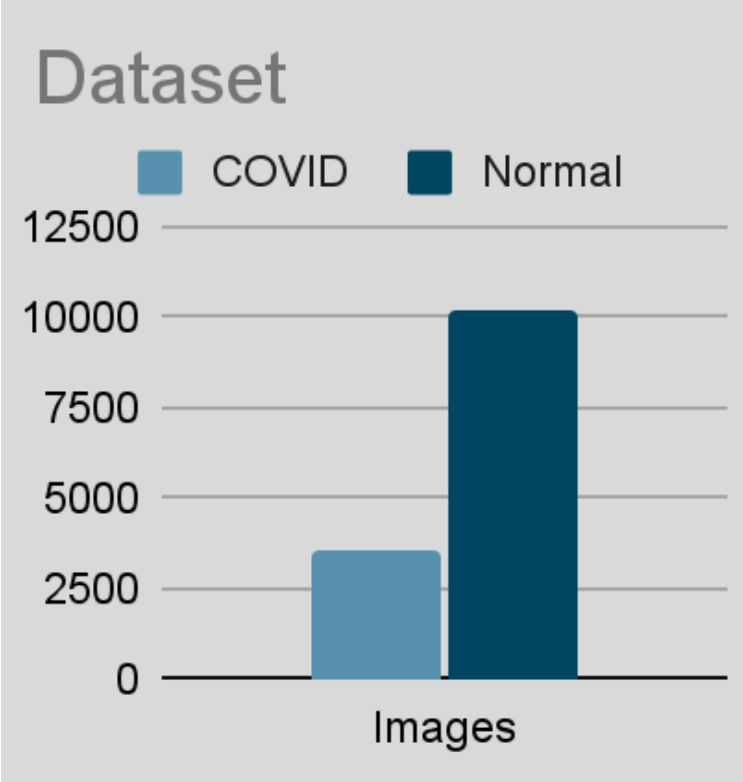


Figure 1. Dataset - Class distribution

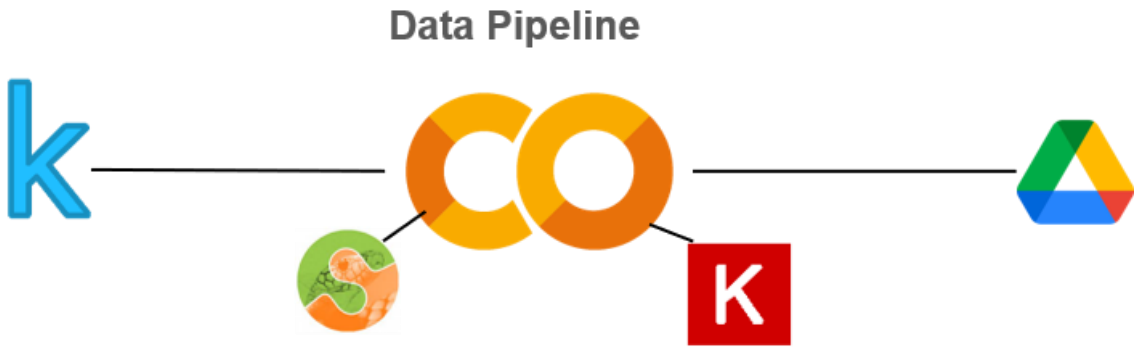


Figure 2. Data pipeline

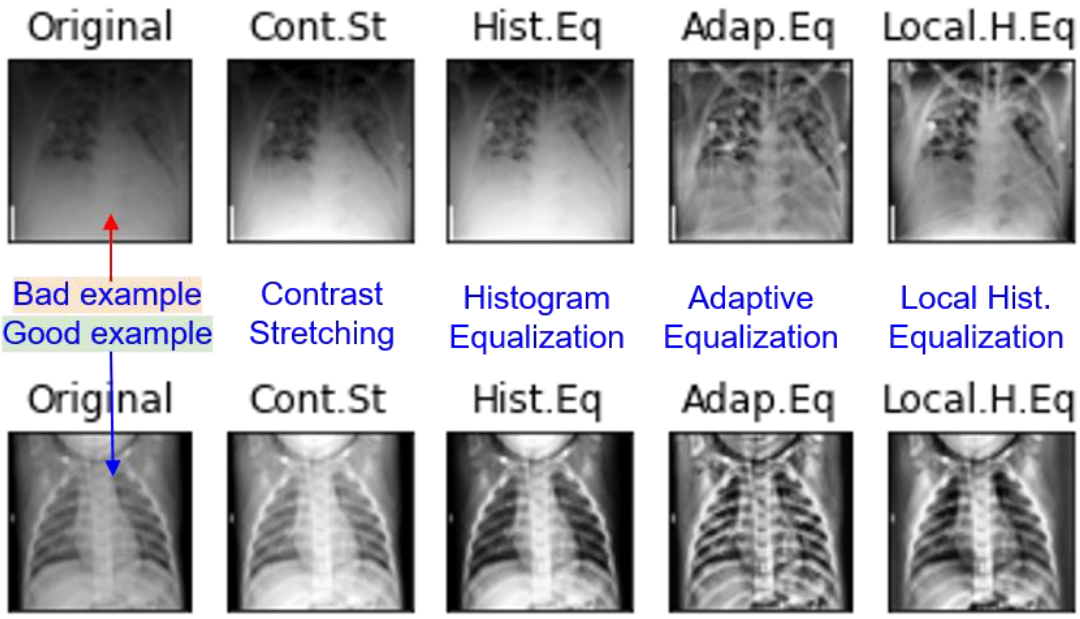


Figure 3. Image pre-processing using scikit-image

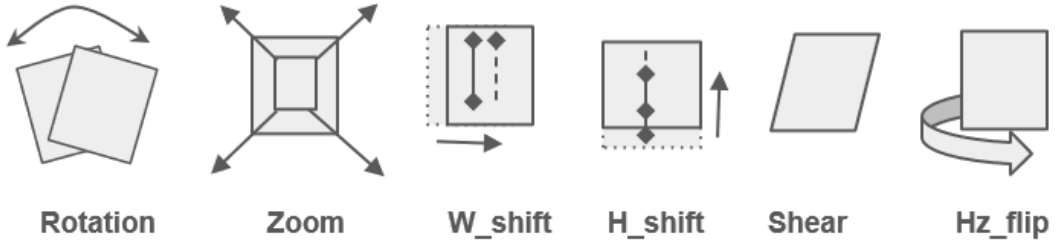


Figure 4. Data Augmentation using Keras image data generator

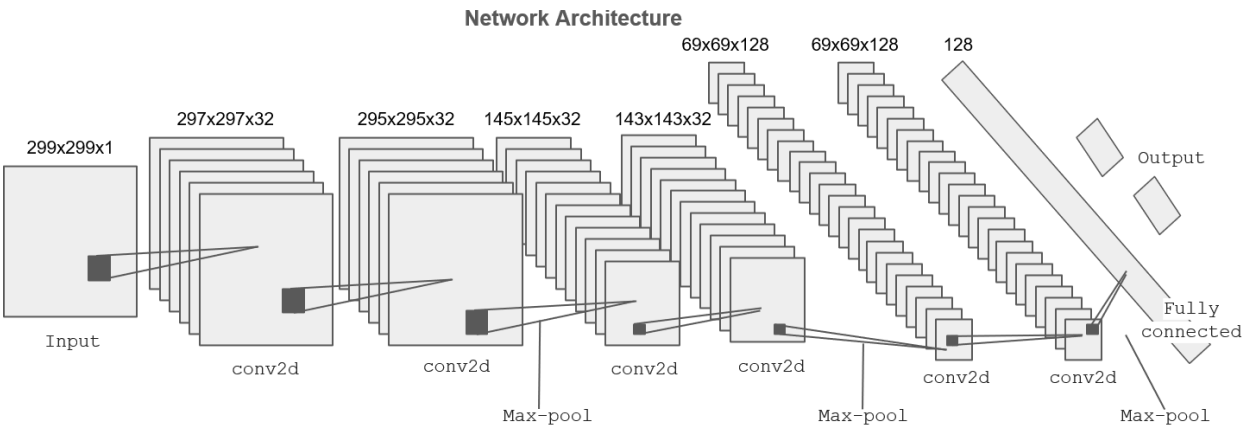


Figure 5. CNN Architecture

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 297, 297, 32)	320
conv2d_1 (Conv2D)	(None, 295, 295, 32)	9248
batch_normalization (BatchNo	(None, 295, 295, 32)	128
max_pooling2d (MaxPooling2D)	(None, 147, 147, 32)	0
conv2d_2 (Conv2D)	(None, 145, 145, 64)	18496
conv2d_3 (Conv2D)	(None, 143, 143, 64)	36928
batch_normalization_1 (Batch	(None, 143, 143, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 71, 71, 64)	0
conv2d_4 (Conv2D)	(None, 69, 69, 128)	73856
conv2d_5 (Conv2D)	(None, 67, 67, 128)	147584
batch_normalization_2 (Batch	(None, 67, 67, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 33, 33, 128)	0
flatten (Flatten)	(None, 139392)	0
dense (Dense)	(None, 128)	17842304
dense_1 (Dense)	(None, 2)	258

Total params: 18,129,890

Figure 6. CNN model summary

```
Epoch 37/100
9/9 - 61s - loss: 0.6260 - accuracy: 0.7465 - val_loss: 0.5180 - val_accuracy: 0.7891
Epoch 38/100
9/9 - 53s - loss: 0.5824 - accuracy: 0.7326 - val_loss: 0.5426 - val_accuracy: 0.7656
Epoch 39/100
9/9 - 60s - loss: 0.6080 - accuracy: 0.7083 - val_loss: 0.5955 - val_accuracy: 0.7188
Epoch 40/100
9/9 - 52s - loss: 0.5519 - accuracy: 0.7604 - val_loss: 0.5871 - val_accuracy: 0.7266
Epoch 41/100
9/9 - 46s - loss: 0.5804 - accuracy: 0.7361 - val_loss: 0.5879 - val_accuracy: 0.7266
Epoch 42/100
9/9 - 49s - loss: 0.5450 - accuracy: 0.7674 - val_loss: 0.6834 - val_accuracy: 0.7266
Epoch 43/100
9/9 - 47s - loss: 0.5663 - accuracy: 0.7465 - val_loss: 0.5348 - val_accuracy: 0.7734
Epoch 44/100
9/9 - 42s - loss: 0.5912 - accuracy: 0.7222 - val_loss: 0.7126 - val_accuracy: 0.7031
Epoch 45/100
9/9 - 48s - loss: 0.5919 - accuracy: 0.7222 - val_loss: 0.5789 - val_accuracy: 0.7344
Epoch 46/100
9/9 - 47s - loss: 0.5429 - accuracy: 0.7674 - val_loss: 0.5714 - val_accuracy: 0.7422
Epoch 47/100
9/9 - 41s - loss: 0.5504 - accuracy: 0.7604 - val_loss: 0.5874 - val_accuracy: 0.7266
```

Figure 7. Training log – Early Stopping

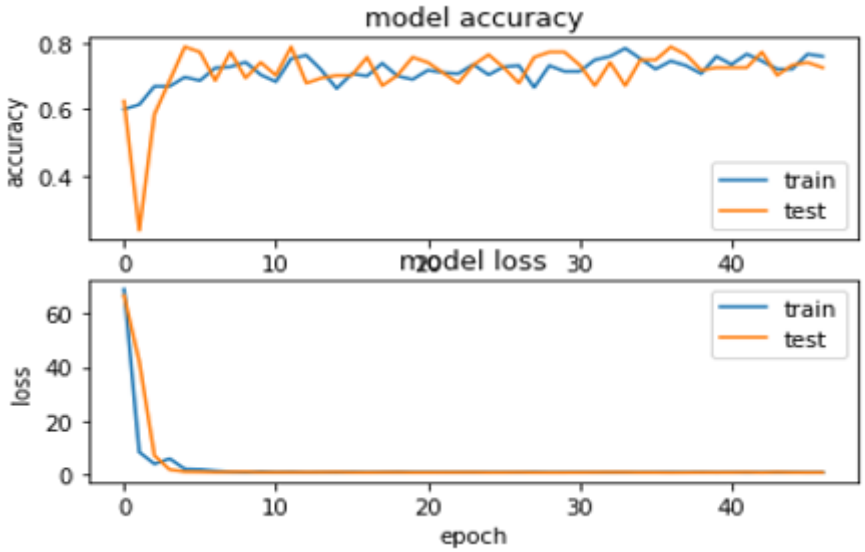


Figure 8. Training - Model learning curve

```
Model: "functional_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
global_max_pooling2d_3 (Glob (None, 512)               0
_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0
=================================================================
Total params: 20,024,384
Trainable params: 0
Non-trainable params: 20,024,384
_________________________________________________________________
```

Figure 9. Reference model - Summary of VGG19 used in the background

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model (Functional)           (None, 512)               20024384
_________________________________________________________________
dense (Dense)                (None, 128)               65664
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 20,090,306
Trainable params: 65,922
Non-trainable params: 20,024,384
_________________________________________________________________
```

Figure 10. Reference Model - Summary with untrainable VGG19

```
Epoch 28/100
9/9 - 136s - loss: 0.3944 - accuracy: 0.8647 - val_loss: 0.2355 - val_accuracy: 0.8984
Epoch 29/100
9/9 - 131s - loss: 0.3224 - accuracy: 0.8889 - val_loss: 0.2830 - val_accuracy: 0.8594
Epoch 30/100
9/9 - 137s - loss: 0.3216 - accuracy: 0.8576 - val_loss: 0.3658 - val_accuracy: 0.8594
Epoch 31/100
9/9 - 119s - loss: 0.4742 - accuracy: 0.8125 - val_loss: 0.2673 - val_accuracy: 0.8828
Epoch 32/100
9/9 - 117s - loss: 0.3636 - accuracy: 0.8576 - val_loss: 0.3485 - val_accuracy: 0.8750
Epoch 33/100
9/9 - 125s - loss: 0.3489 - accuracy: 0.8472 - val_loss: 0.4496 - val_accuracy: 0.8125
Epoch 34/100
9/9 - 110s - loss: 0.3378 - accuracy: 0.8472 - val_loss: 0.2952 - val_accuracy: 0.9141
Epoch 35/100
9/9 - 116s - loss: 0.3548 - accuracy: 0.8368 - val_loss: 0.3977 - val_accuracy: 0.8828
Epoch 36/100
9/9 - 119s - loss: 0.3742 - accuracy: 0.8368 - val_loss: 0.3220 - val_accuracy: 0.8828
Epoch 37/100
9/9 - 108s - loss: 0.3331 - accuracy: 0.8542 - val_loss: 0.2928 - val_accuracy: 0.8594
Epoch 38/100
9/9 - 101s - loss: 0.4664 - accuracy: 0.7744 - val_loss: 0.3407 - val_accuracy: 0.8750
```

Figure 11. Reference Model - Training log

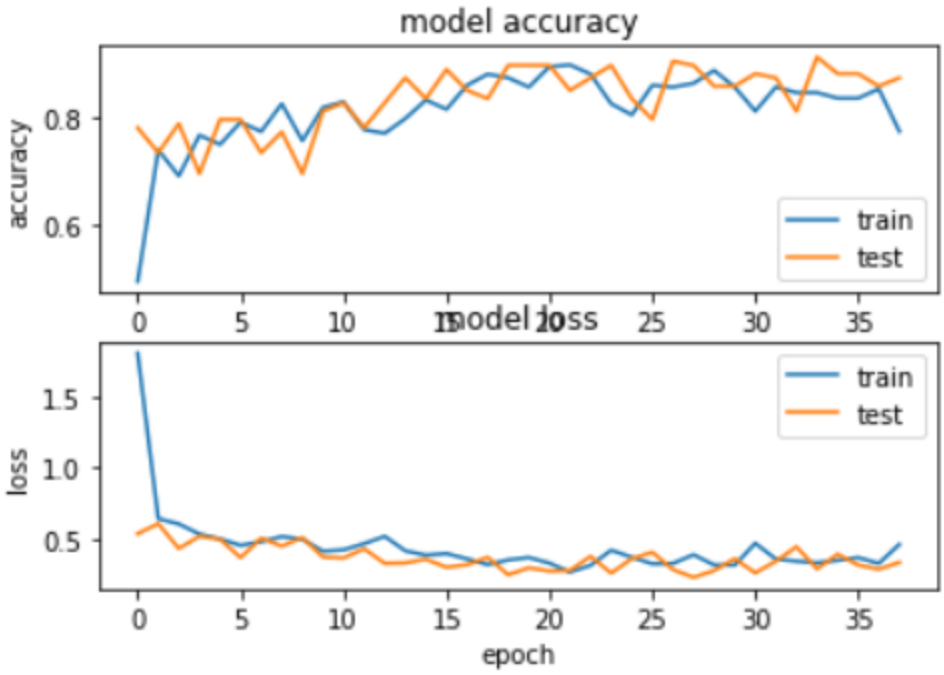


Figure 12. Reference Model - Learning curve

Final code - Project - ANLY 535

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
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, AvgPool2D,MaxPool2D
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.optimizers import Adam, SGD
import tensorflow as tf
from tensorflow.keras.preprocessing import image
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [ ]:
```
path='/content/gdrive/MyDrive/COVID-19_Radiography_Dataset'
savepath = '/content/gdrive/MyDrive/COVID-19_Radiography_Dataset_pp'
```

In [ ]:
```
from skimage.filters.rank import equalize
from skimage.morphology import disk
import os
import skimage.io as io
import skimage.color as color
import skimage.transform as trf
import numpy as np
for directory in sorted(os.listdir(path)):
        data_path = path +"/"+ directory
        save_path = savepath +"/"+ directory
        for im in os.listdir(data_path)[:]:
            image = io.imread(f"{data_path}/{im}")
            #image - np.array(image)
            image = equalize(image, disk(70))
            #image = color.rgb2gray(image)
            #image = trf.resize(image, (image_sz, image_sz))
            #images.append(image)
            #labels.append(directory)
            io.imsave(f"{save_path}/{im}", image)
```

In [ ]:
```
image_prepro_path = '/content/gdrive/MyDrive/COVID-19_Radiography_Dataset_pp'
classes=["COVID",  "Normal"]
num_classes = len(classes)
batch_size=32
target_size=(299, 299)
# Splitting the dataset into 70:30 by ImageGenerator
data_gen = ImageDataGenerator(rotation_range=15, zoom_range=0.2, width_shift_range=0.2, height_shift_range=0.2, shear_rar

# Loading images to train generator
train_set = data_gen.flow_from_directory(directory=image_prepro_path,
                                         target_size=target_size,
                                         class_mode='categorical',
                                         subset='training',
                                         shuffle=True, classes=classes,
                                         batch_size=batch_size,
                                         color_mode="grayscale")
# Loading images to test generator
test_set = data_gen.flow_from_directory(directory=image_prepro_path,
                                         target_size=target_size,
                                         class_mode='categorical',
                                         subset='validation',
                                         shuffle=True, classes=classes,
                                         batch_size=batch_size,
                                         color_mode="grayscale")
```

```
Found 9674 images belonging to 2 classes.
Found 4144 images belonging to 2 classes.
```

In [ ]:
```
def covid_model():

    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(299, 299, 1)))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(BatchNormalization(momentum=0.9))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(BatchNormalization(momentum=0.9))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(BatchNormalization(momentum=0.9))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
   # model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(128, activation='relu'))
```

```
    model.add(Dense(num_classes, activation='softmax'))
    # compile model
    #optimizer_sgd = SGD(lr=0.01, momentum=0.9)
    opt=Adam(learning_rate=0.008)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=["accuracy"])
    model.summary()
    return model
```

In [ ]:
```
#checkpoint
import tensorflow
from tensorflow.keras.callbacks import ModelCheckpoint
early_stop = tensorflow.keras.callbacks.EarlyStopping(monitor='val_loss', patience= 10)
callbacks_list= ModelCheckpoint('/content/gdrive/MyDrive/ANLY535_Models/Weights-{epoch:03d}--{val_loss:.5f}.hdf5', monitc
callbacks = [early_stop, callbacks_list]
```

In [ ]:
```
covidmodel = covid_model()
# fitting the model
history = covidmodel.fit_generator(train_set, steps_per_epoch=len(train_set) // batch_size, validation_steps=len(test_set
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 297, 297, 32)      320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 295, 295, 32)      9248
_________________________________________________________________
batch_normalization (BatchNo (None, 295, 295, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 147, 147, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 145, 145, 64)      18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 143, 143, 64)      36928
_________________________________________________________________
batch_normalization_1 (Batch (None, 143, 143, 64)      256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 71, 71, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 69, 69, 128)       73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 67, 67, 128)       147584
_________________________________________________________________
batch_normalization_2 (Batch (None, 67, 67, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 33, 33, 128)       0
_________________________________________________________________
```

```
flatten (Flatten)            (None, 139392)            0         
_________________________________________________________________
dense (Dense)                (None, 128)               17842304  
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258       
=================================================================
Total params: 18,129,890
Trainable params: 18,129,442
Non-trainable params: 448
_________________________________________________________________
/usr/local/lib/python3.7/dist-packages/keras/engine/training.py:1915: UserWarning: `Model.fit_generator` is deprecated an
d will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/100
9/9 - 231s - loss: 69.1473 - accuracy: 0.6007 - val_loss: 66.7738 - val_accuracy: 0.6250
Epoch 2/100
9/9 - 176s - loss: 8.2342 - accuracy: 0.6146 - val_loss: 42.0559 - val_accuracy: 0.2344
Epoch 3/100
9/9 - 174s - loss: 3.7898 - accuracy: 0.6701 - val_loss: 6.9383 - val_accuracy: 0.5859
Epoch 4/100
9/9 - 162s - loss: 5.7808 - accuracy: 0.6701 - val_loss: 1.6611 - val_accuracy: 0.6875
Epoch 5/100
9/9 - 155s - loss: 1.8932 - accuracy: 0.6979 - val_loss: 0.8715 - val_accuracy: 0.7891
Epoch 6/100
9/9 - 161s - loss: 1.7165 - accuracy: 0.6875 - val_loss: 0.8035 - val_accuracy: 0.7734
Epoch 7/100
9/9 - 158s - loss: 1.2044 - accuracy: 0.7257 - val_loss: 0.6898 - val_accuracy: 0.6875
Epoch 8/100
9/9 - 153s - loss: 0.8640 - accuracy: 0.7292 - val_loss: 0.7870 - val_accuracy: 0.7734
Epoch 9/100
9/9 - 144s - loss: 0.6553 - accuracy: 0.7431 - val_loss: 0.8317 - val_accuracy: 0.6953
Epoch 10/100
9/9 - 131s - loss: 0.9062 - accuracy: 0.7049 - val_loss: 0.6439 - val_accuracy: 0.7422
Epoch 11/100
9/9 - 136s - loss: 0.6554 - accuracy: 0.6840 - val_loss: 0.6541 - val_accuracy: 0.7031
Epoch 12/100
9/9 - 129s - loss: 0.7534 - accuracy: 0.7535 - val_loss: 0.6643 - val_accuracy: 0.7891
Epoch 13/100
9/9 - 133s - loss: 0.5671 - accuracy: 0.7639 - val_loss: 0.6682 - val_accuracy: 0.6797
Epoch 14/100
9/9 - 120s - loss: 0.6372 - accuracy: 0.7188 - val_loss: 0.6886 - val_accuracy: 0.6953
Epoch 15/100
9/9 - 122s - loss: 0.6424 - accuracy: 0.6632 - val_loss: 0.7424 - val_accuracy: 0.7031
Epoch 16/100
9/9 - 110s - loss: 0.7041 - accuracy: 0.7083 - val_loss: 0.6075 - val_accuracy: 0.7031
Epoch 17/100
9/9 - 111s - loss: 0.6063 - accuracy: 0.7014 - val_loss: 0.5735 - val_accuracy: 0.7578
Epoch 18/100
9/9 - 102s - loss: 0.6522 - accuracy: 0.7396 - val_loss: 0.6335 - val_accuracy: 0.6719
```

```
Epoch 19/100
9/9 - 111s - loss: 0.7350 - accuracy: 0.7014 - val_loss: 0.6588 - val_accuracy: 0.7031
Epoch 20/100
9/9 - 100s - loss: 0.6394 - accuracy: 0.6910 - val_loss: 0.5658 - val_accuracy: 0.7578
Epoch 21/100
9/9 - 105s - loss: 0.6352 - accuracy: 0.7188 - val_loss: 0.5705 - val_accuracy: 0.7422
Epoch 22/100
9/9 - 91s - loss: 0.5974 - accuracy: 0.7118 - val_loss: 0.6001 - val_accuracy: 0.7109
Epoch 23/100
9/9 - 82s - loss: 0.6043 - accuracy: 0.7083 - val_loss: 0.6276 - val_accuracy: 0.6797
Epoch 24/100
9/9 - 91s - loss: 0.6069 - accuracy: 0.7361 - val_loss: 0.6506 - val_accuracy: 0.7344
Epoch 25/100
9/9 - 79s - loss: 0.6074 - accuracy: 0.7049 - val_loss: 0.5516 - val_accuracy: 0.7656
Epoch 26/100
9/9 - 80s - loss: 0.5855 - accuracy: 0.7292 - val_loss: 0.5834 - val_accuracy: 0.7266
Epoch 27/100
9/9 - 80s - loss: 0.5811 - accuracy: 0.7326 - val_loss: 0.6480 - val_accuracy: 0.6797
Epoch 28/100
9/9 - 72s - loss: 0.6588 - accuracy: 0.6667 - val_loss: 0.5577 - val_accuracy: 0.7578
Epoch 29/100
9/9 - 70s - loss: 0.5841 - accuracy: 0.7326 - val_loss: 0.5430 - val_accuracy: 0.7734
Epoch 30/100
9/9 - 75s - loss: 0.5983 - accuracy: 0.7153 - val_loss: 0.5435 - val_accuracy: 0.7734
Epoch 31/100
9/9 - 72s - loss: 0.5929 - accuracy: 0.7153 - val_loss: 0.5788 - val_accuracy: 0.7344
Epoch 32/100
9/9 - 79s - loss: 0.5638 - accuracy: 0.7500 - val_loss: 0.6344 - val_accuracy: 0.6719
Epoch 33/100
9/9 - 77s - loss: 0.5751 - accuracy: 0.7604 - val_loss: 0.6815 - val_accuracy: 0.7422
Epoch 34/100
9/9 - 54s - loss: 0.5690 - accuracy: 0.7847 - val_loss: 0.6685 - val_accuracy: 0.6719
Epoch 35/100
9/9 - 58s - loss: 0.5628 - accuracy: 0.7535 - val_loss: 0.5768 - val_accuracy: 0.7500
Epoch 36/100
9/9 - 57s - loss: 0.5919 - accuracy: 0.7222 - val_loss: 0.5623 - val_accuracy: 0.7500
Epoch 37/100
9/9 - 61s - loss: 0.6260 - accuracy: 0.7465 - val_loss: 0.5180 - val_accuracy: 0.7891
Epoch 38/100
9/9 - 53s - loss: 0.5824 - accuracy: 0.7326 - val_loss: 0.5426 - val_accuracy: 0.7656
Epoch 39/100
9/9 - 60s - loss: 0.6080 - accuracy: 0.7083 - val_loss: 0.5955 - val_accuracy: 0.7188
Epoch 40/100
9/9 - 52s - loss: 0.5519 - accuracy: 0.7604 - val_loss: 0.5871 - val_accuracy: 0.7266
Epoch 41/100
9/9 - 46s - loss: 0.5804 - accuracy: 0.7361 - val_loss: 0.5879 - val_accuracy: 0.7266
Epoch 42/100
9/9 - 49s - loss: 0.5450 - accuracy: 0.7674 - val_loss: 0.6834 - val_accuracy: 0.7266
Epoch 43/100
```

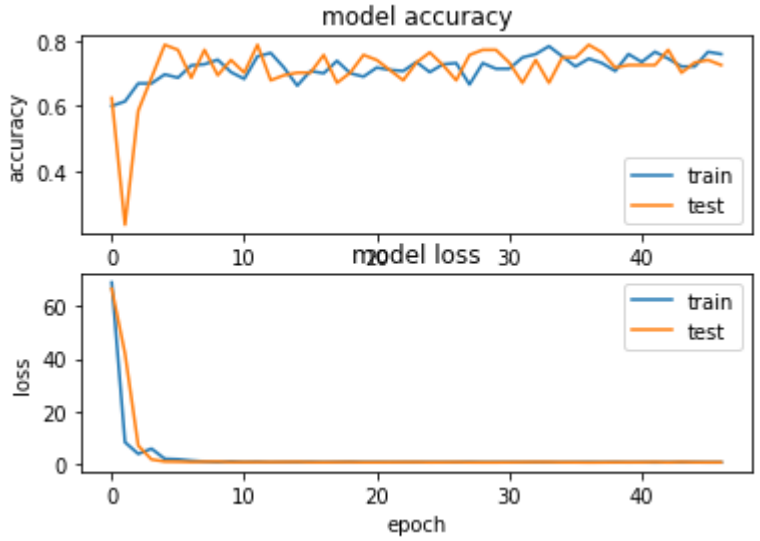
```
9/9 - 47s - loss: 0.5663 - accuracy: 0.7465 - val_loss: 0.5348 - val_accuracy: 0.7734
Epoch 44/100
9/9 - 42s - loss: 0.5912 - accuracy: 0.7222 - val_loss: 0.7126 - val_accuracy: 0.7031
Epoch 45/100
9/9 - 48s - loss: 0.5919 - accuracy: 0.7222 - val_loss: 0.5789 - val_accuracy: 0.7344
Epoch 46/100
9/9 - 47s - loss: 0.5429 - accuracy: 0.7674 - val_loss: 0.5714 - val_accuracy: 0.7422
Epoch 47/100
9/9 - 41s - loss: 0.5504 - accuracy: 0.7604 - val_loss: 0.5874 - val_accuracy: 0.7266
```

In [ ]:

```
plt.subplot(2,1,1)
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='lower right')

plt.subplot(2,1,2)
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 right')

plt.show()
```



Reference model - VGG19

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img, img_to_array
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D,GlobalAveragePooling2D, Conv3D, AveragePooling2D
from tensorflow.keras.layers import Activation, Dropout, BatchNormalization, Flatten, Dense, AvgPool2D,MaxPool2D
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.optimizers import Adam, SGD
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.vgg19 import preprocess_input
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [ ]:
```
# path='/content/gdrive/MyDrive/COVID-19_Radiography_Dataset'
# savepath = '/content/gdrive/MyDrive/COVID-19_Radiography_Dataset_pp'
```

In [ ]:
```
# from skimage.filters.rank import equalize
# from skimage.morphology import disk
# import os
# import skimage.io as io
# import skimage.color as color
# import skimage.transform as trf
# import numpy as np
# for directory in sorted(os.listdir(path)):
#         data_path = path +"/"+ directory
#         save_path = savepath +"/"+ directory
#         for im in os.listdir(data_path)[:]:
#             image = io.imread(f"{data_path}/{im}")
#             #image - np.array(image)
#             image = equalize(image, disk(70))
#             #image = color.rgb2gray(image)
#             #image = trf.resize(image, (image_sz, image_sz))
#             #images.append(image)
```

```
#             #labels.append(directory)
#             io.imsave(f"{save_path}/{im}", image)
```

In [ ]:
```
image_prepro_path = '/content/gdrive/MyDrive/COVID-19_Radiography_Dataset_pp'
classes=["COVID",  "Normal"]
num_classes = len(classes)
batch_size=32
target_size=(224, 224)
# Splitting the dataset into 70:30 by ImageGenerator
data_gen = ImageDataGenerator(preprocessing_function=preprocess_input,rotation_range=15, zoom_range=0.2, width_shift_rang

# Loading images to train generator
train_set = data_gen.flow_from_directory(directory=image_prepro_path,
                                         target_size=target_size,
                                         class_mode='categorical',
                                         subset='training',
                                         shuffle=True, classes=classes,
                                         batch_size=batch_size,
                                         color_mode="rgb")
# Loading images to test generator
test_set = data_gen.flow_from_directory(directory=image_prepro_path,
                                         target_size=target_size,
                                         class_mode='categorical',
                                         subset='validation',
                                         shuffle=True, classes=classes,
                                         batch_size=batch_size,
                                         color_mode="rgb")
```

```
Found 9674 images belonging to 2 classes.
Found 4144 images belonging to 2 classes.
```

In [ ]:
```
#checkpoint
import tensorflow
from tensorflow.keras.callbacks import ModelCheckpoint
early_stop = tensorflow.keras.callbacks.EarlyStopping(monitor='val_loss', patience= 10)
callbacks_list= ModelCheckpoint('/content/gdrive/MyDrive/ANLY535_Models/Weights-{epoch:03d}--{val_loss:.5f}.hdf5', monitc
callbacks = [early_stop, callbacks_list]
```

In [ ]:
```
from tensorflow.keras.models import Model
import tensorflow.keras as keras

vgg = VGG19(include_top=False, weights='imagenet', input_shape=(224,224,3),pooling='max')
output = vgg.layers[-1].output
output = tensorflow.keras.layers.Flatten()(output)
vgg = Model(vgg.input, output)
```

```
for layer in vgg.layers:
    layer.trainable = False

model = Sequential()
model.add(vgg)
model.add(Dense(128, activation='relu'))
model.add(Dense(2, activation='softmax'))
model.summary()

opt=Adam(learning_rate=0.008)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=["accuracy"])
#model.evaluate_generator(test_set)
# # fitting the model
history1 = model.fit_generator(train_set, steps_per_epoch=len(train_set) // batch_size, validation_steps=len(test_set) //
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf
_kernels_notop.h5
80142336/80134624 [==============================] - 2s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model (Functional)           (None, 512)               20024384
_________________________________________________________________
dense (Dense)                (None, 128)               65664
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 20,090,306
Trainable params: 65,922
Non-trainable params: 20,024,384
_________________________________________________________________
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator
` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/100
9/9 - 376s - loss: 1.8034 - accuracy: 0.4931 - val_loss: 0.5413 - val_accuracy: 0.7812
Epoch 2/100
9/9 - 342s - loss: 0.6463 - accuracy: 0.7396 - val_loss: 0.6111 - val_accuracy: 0.7344
Epoch 3/100
9/9 - 337s - loss: 0.6089 - accuracy: 0.6910 - val_loss: 0.4376 - val_accuracy: 0.7891
Epoch 4/100
9/9 - 328s - loss: 0.5388 - accuracy: 0.7674 - val_loss: 0.5233 - val_accuracy: 0.6953
Epoch 5/100
9/9 - 309s - loss: 0.5045 - accuracy: 0.7500 - val_loss: 0.5024 - val_accuracy: 0.7969
Epoch 6/100
9/9 - 310s - loss: 0.4591 - accuracy: 0.7917 - val_loss: 0.3737 - val_accuracy: 0.7969
```

```
Epoch 7/100
9/9 - 292s - loss: 0.4851 - accuracy: 0.7743 - val_loss: 0.5100 - val_accuracy: 0.7344
Epoch 8/100
9/9 - 272s - loss: 0.5211 - accuracy: 0.8264 - val_loss: 0.4539 - val_accuracy: 0.7734
Epoch 9/100
9/9 - 258s - loss: 0.5026 - accuracy: 0.7569 - val_loss: 0.5146 - val_accuracy: 0.6953
Epoch 10/100
9/9 - 263s - loss: 0.4186 - accuracy: 0.8194 - val_loss: 0.3779 - val_accuracy: 0.8125
Epoch 11/100
9/9 - 238s - loss: 0.4285 - accuracy: 0.8299 - val_loss: 0.3711 - val_accuracy: 0.8281
Epoch 12/100
9/9 - 244s - loss: 0.4712 - accuracy: 0.7778 - val_loss: 0.4357 - val_accuracy: 0.7812
Epoch 13/100
9/9 - 229s - loss: 0.5227 - accuracy: 0.7708 - val_loss: 0.3338 - val_accuracy: 0.8281
Epoch 14/100
9/9 - 237s - loss: 0.4213 - accuracy: 0.7986 - val_loss: 0.3370 - val_accuracy: 0.8750
Epoch 15/100
9/9 - 210s - loss: 0.3910 - accuracy: 0.8333 - val_loss: 0.3625 - val_accuracy: 0.8359
Epoch 16/100
9/9 - 201s - loss: 0.4021 - accuracy: 0.8160 - val_loss: 0.3064 - val_accuracy: 0.8906
Epoch 17/100
9/9 - 201s - loss: 0.3653 - accuracy: 0.8611 - val_loss: 0.3239 - val_accuracy: 0.8516
Epoch 18/100
9/9 - 200s - loss: 0.3253 - accuracy: 0.8819 - val_loss: 0.3762 - val_accuracy: 0.8359
Epoch 19/100
9/9 - 180s - loss: 0.3577 - accuracy: 0.8750 - val_loss: 0.2538 - val_accuracy: 0.8984
Epoch 20/100
9/9 - 190s - loss: 0.3747 - accuracy: 0.8576 - val_loss: 0.3028 - val_accuracy: 0.8984
Epoch 21/100
9/9 - 175s - loss: 0.3354 - accuracy: 0.8958 - val_loss: 0.2798 - val_accuracy: 0.8984
Epoch 22/100
9/9 - 178s - loss: 0.2731 - accuracy: 0.8993 - val_loss: 0.2843 - val_accuracy: 0.8516
Epoch 23/100
9/9 - 183s - loss: 0.3195 - accuracy: 0.8819 - val_loss: 0.3826 - val_accuracy: 0.8750
Epoch 24/100
9/9 - 157s - loss: 0.4259 - accuracy: 0.8264 - val_loss: 0.2658 - val_accuracy: 0.8984
Epoch 25/100
9/9 - 160s - loss: 0.3766 - accuracy: 0.8056 - val_loss: 0.3660 - val_accuracy: 0.8359
Epoch 26/100
9/9 - 153s - loss: 0.3309 - accuracy: 0.8611 - val_loss: 0.4116 - val_accuracy: 0.7969
Epoch 27/100
9/9 - 142s - loss: 0.3330 - accuracy: 0.8576 - val_loss: 0.2903 - val_accuracy: 0.9062
Epoch 28/100
9/9 - 136s - loss: 0.3944 - accuracy: 0.8647 - val_loss: 0.2355 - val_accuracy: 0.8984
Epoch 29/100
9/9 - 131s - loss: 0.3224 - accuracy: 0.8889 - val_loss: 0.2830 - val_accuracy: 0.8594
Epoch 30/100
9/9 - 137s - loss: 0.3216 - accuracy: 0.8576 - val_loss: 0.3658 - val_accuracy: 0.8594
Epoch 31/100
```

```
9/9 - 119s - loss: 0.4742 - accuracy: 0.8125 - val_loss: 0.2673 - val_accuracy: 0.8828
Epoch 32/100
9/9 - 117s - loss: 0.3636 - accuracy: 0.8576 - val_loss: 0.3485 - val_accuracy: 0.8750
Epoch 33/100
9/9 - 125s - loss: 0.3489 - accuracy: 0.8472 - val_loss: 0.4496 - val_accuracy: 0.8125
Epoch 34/100
9/9 - 110s - loss: 0.3378 - accuracy: 0.8472 - val_loss: 0.2952 - val_accuracy: 0.9141
Epoch 35/100
9/9 - 116s - loss: 0.3548 - accuracy: 0.8368 - val_loss: 0.3977 - val_accuracy: 0.8828
Epoch 36/100
9/9 - 119s - loss: 0.3742 - accuracy: 0.8368 - val_loss: 0.3220 - val_accuracy: 0.8828
Epoch 37/100
9/9 - 108s - loss: 0.3331 - accuracy: 0.8542 - val_loss: 0.2928 - val_accuracy: 0.8594
Epoch 38/100
9/9 - 101s - loss: 0.4664 - accuracy: 0.7744 - val_loss: 0.3407 - val_accuracy: 0.8750
```

In [ ]:

```
def covid_model():

    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(299, 299, 1)))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(BatchNormalization(momentum=0.9))
    model.add(AveragePooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(BatchNormalization(momentum=0.9))
    model.add(AveragePooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(BatchNormalization(momentum=0.9))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
   # model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # compile model
    #optimizer_sgd = SGD(lr=0.01, momentum=0.9)
    opt=Adam(learning_rate=0.008)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=["accuracy"])
    model.summary()
    return model
```

In [ ]:

```
# covidmodel = covid_model()
# # fitting the model
# history = covidmodel.fit_generator(train_set, steps_per_epoch=len(train_set) // batch_size, validation_steps=len(test_s
```

In [ ]:
```
plt.subplot(2,1,1)
plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')

plt.subplot(2,1,2)
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')

plt.show()
```