

# Detecting COVID-19 From Chest X-Rays Using Convolution Neural Networks and transfer learning deep learning techniques

## Project Description:

### Introduction

COVID-19, caused by the SARS-CoV-2 virus, has emerged as a global pandemic, impacting millions of lives worldwide. Early and accurate diagnosis is crucial for effective treatment and containment of the disease. Chest X-ray and computed tomography (CT) scans have been widely used for COVID-19 diagnosis, providing valuable insights into lung abnormalities associated with the infection.

### Project Aim

This project aims to develop a deep learning-based approach using convolutional neural networks (CNNs) and transfer learning techniques to accurately detect COVID-19 from chest X-ray and CT scans.

### Expected Outcomes

The expected outcomes of this project include:

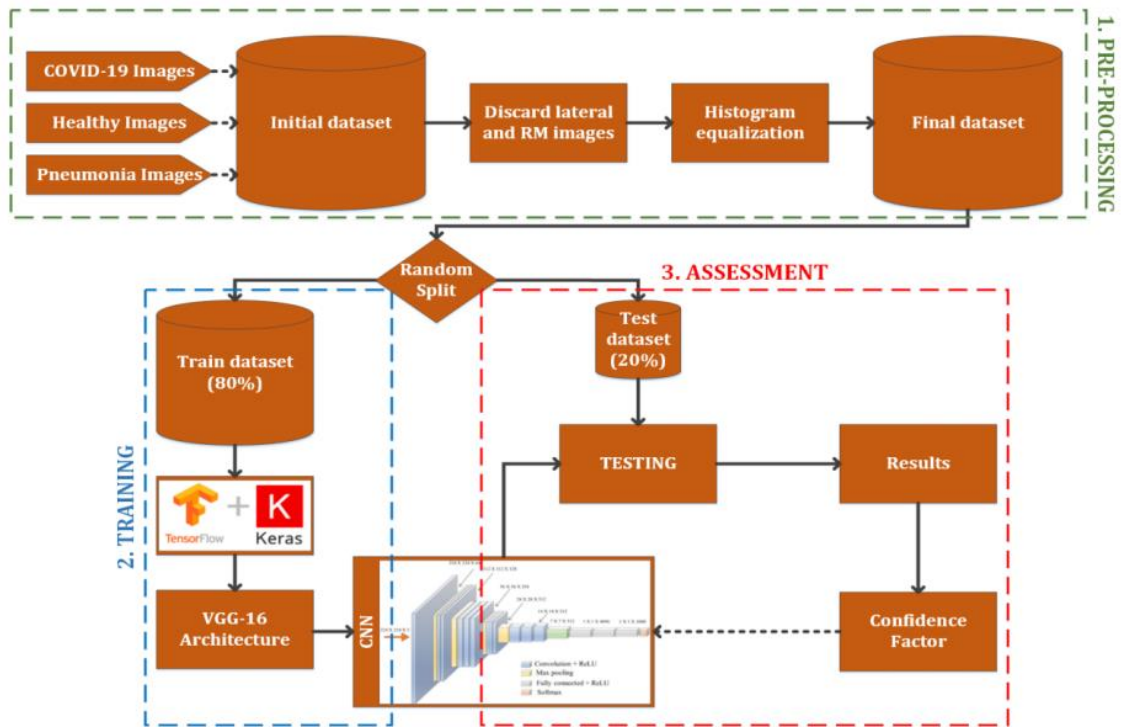
- **Accurate COVID-19 Detection:** Development of highly accurate CNN models capable of detecting COVID-19 from chest X-ray and CT scans with high sensitivity and specificity.
- **Reduced Diagnostic Burden:** Automation of COVID-19 detection using CNNs, reducing the workload on radiologists and expediting patient diagnosis.
- **Improved Diagnostic Accuracy:** Enhancement of diagnostic accuracy compared to traditional methods, potentially leading to better patient outcomes.

### Significance and Impact

The successful development of CNN-based models for COVID-19 detection can significantly impact healthcare by:

**Early Diagnosis and Treatment:** Enabling early identification of COVID-19 cases, facilitating prompt treatment and improving patient outcomes.

## Technical Architecture:



## Prerequisites:

To complete this project, you must require the following software's, concepts and packages

- **Anaconda navigator or google colab**
  - In google colab is the better option for doing deep learning projects for its pre installed packages for implementing or building the model of cnn and transfer learning
- **Python packages:**
  - Type “pip install numpy” and click enter.
  - Type “pip install pandas” and click enter..
  - Type “pip install tensorflow==2.3.2” and click enter.
  - Type “pip install keras==2.3.1” and click enter.
  - Type “pip install Flask” and click enter.

## Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- **Deep Learning Concepts**
- **CNN:** <https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add>
- **VGG16:**

<https://medium.com/@mygreatlearning/what-is-vgg16-introduction-to-vgg16-f2d63849f615>

- **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.  
Link: [https://www.youtube.com/watch?v=Ij4J\\_CvBnt0](https://www.youtube.com/watch?v=Ij4J_CvBnt0)

## **Project Objectives:**

By the end of this project you'll understand:

- Preprocessing the images.
- Applying Transfer learning algorithms on the dataset.
- How deep neural networks detect the disease.
- You will be able to know how to find the accuracy of the model.
- You will be able to Build web applications using the Flask framework.
- Doing trial and error methods for greater accuracy for doing predictions
- Rectifying errors while implementing transfer learning and deep learning algorithms
- Testing of differential images and predicting it

## **Project Flow:**

The user interacts with the UI(User Interface) to choose the image.

The chosen image analyzed by the model which is integrated with flask application.

The Xception Model analyzes the image, then the prediction is showcased on the Flask UI.

To accomplish this, we have to complete all the activities and tasks listed below

### **Data Collection and Preprocessing**

- Gather chest X-ray and CT scan datasets of COVID-19 and non-COVID-19 patients from credible sources like kaggle and roboflow or other websites etc
- Upload kaggle dataset into the drive in zip folder format and unzip it in colab by mounting drive and uploading required files into the colab
- Divide the preprocessed dataset into training, validation, and testing sets for model training and evaluation.

### **Data Pre-processing.**

- Import the required library
- Configure ImageDataGenerator class
- Apply ImageDataGenerator functionality to Trainset and Testset

### **Model Architecture Design and Training**

- Choose pre-trained CNN models like VGG16, ResNet50, or InceptionV3 for transfer learning, leveraging their learned features.
- Modify pre-trained models by removing top classification layers and adding new layers specific to COVID-19 detection.
- Optimize hyperparameters like learning rate, optimizer, batch size, and epochs using the validation set to enhance performance.
- Train the CNN models on the training dataset, monitoring training metrics like accuracy and loss to assess learning progress.
- Implement early stopping to prevent overfitting and halt training when validation loss increases or performance plateaus.

### **Model Building**

- Pre-trained CNN model as a Feature Extractor
- Adding Dense Layer
- Configure the Learning Process
- Train the model
- Save the Model
- Test the model

### **Model building for transfer learning**

- Import Libraries
- Load Pre-trained Model
- Freeze Pre-trained Layers:
- Add New Layers
- Compile the Model.
- Prepare the Dataset
- Split the Dataset
- Train the Model
- Evaluate the Model
- Save the Model

## **Model Deployment and Analysis**

Integrate trained CNN models into a user-friendly interface or application for easy access and interpretation.

Analyze trained CNN models to identify the most important features for COVID-19 detection using visualization and feature ablation.

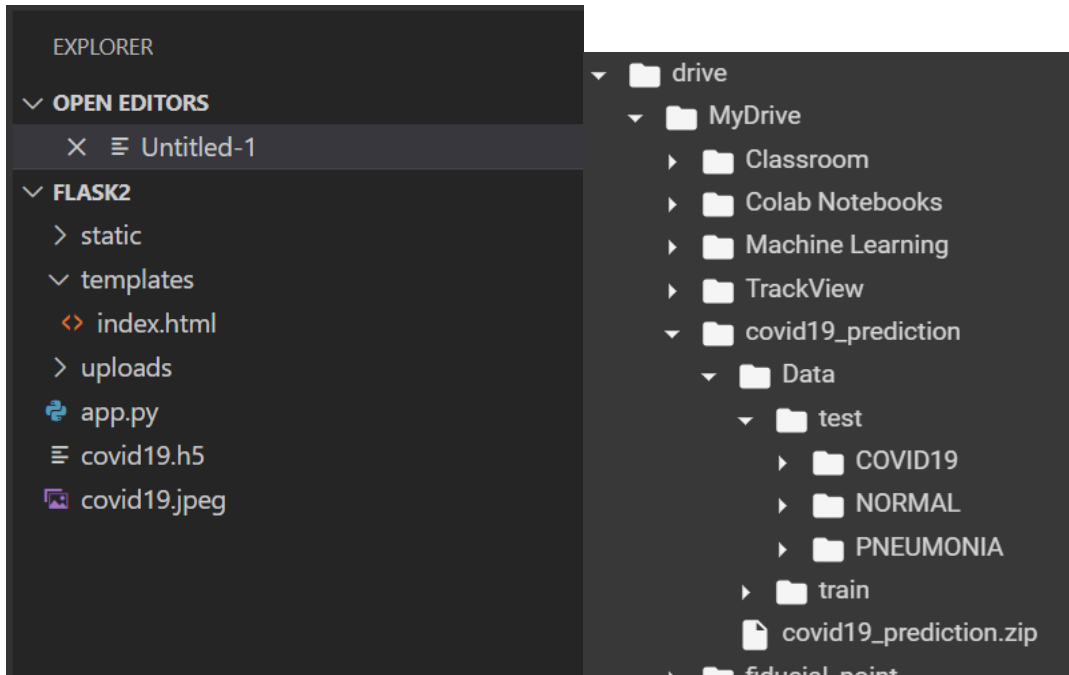
Conduct clinical trials to evaluate the real-world performance of deployed models in a clinical setting, comparing their predictions to expert radiologist interpretations.

## **Application Building**

- Create an HTML file by using bootstrap template (css and js files as well) and create folder which contains model,static,template and upload folder as well which contains some images
- Build Python Code And create flask that rout to our website and create choose option which it connects to upload folder for selecting required images for prediction
- Our html website is ready for prediction which contains lungs scan image choosing option with giving accurate covid19 prediction for the given image

## Project Structure:

Create a Project folder which contains files as shown below



- Flask folder consists of static(css and js files), templates ,app1.py ,some images and model.h5 file
- Covid 19 prediction is a colab ipbny file notebook and contains traning and testing files

### Activity 1: Download the dataset and data collection

Collect images of Covid-19 Chest X-ray images then organized into subdirectories based on their respective names as shown in the project structure. Create folders of types of Covid-19 that need to be recognized.

In this project, we have collected images of 3 types of Covid-19 images like **covid19** , **normal**, **PNEUMONIA** and they are saved in the respective sub directories with their respective names.

You can download the dataset used in this project using the below link

Dataset:-

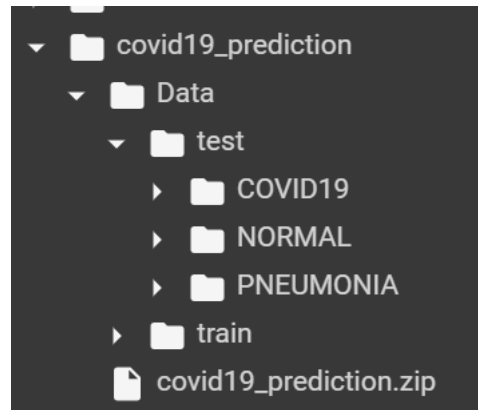
<https://www.kaggle.com/datasets/prashant268/chest-xray-covid19-pneumonia>

Upload the dataset into google drive and connect the google colab with drive using the below code

Once mounted the drive create a folder with project name as **covid19 prediction project** and move the dataset to that folder and mount to that folder and upload it in the colab by connecting the drive in the colab

Mount the drive by using the below command and unzip it

```
+ Code + Text
[ ] import zipfile
zip_path = "/content/drive/MyDrive/covid19_prediction /covid19_prediction.zip" # Replace with the actual path
extract_path = "/content/drive/MyDrive/covid19_prediction " # Replace with the desired path for the extracted
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)
```



The screenshot shows a file explorer interface with the following structure:

- ▼ covid19\_prediction
  - ▼ Data
    - ▼ test
      - ▶ COVID19
      - ▶ NORMAL
      - ▶ PNEUMONIA
    - ▶ train
  - covid19\_prediction.zip

## Activity 2: Create training and testing dataset and data agumentation

Imports ImageDataGenerator for manipulating and augmenting image data

Command is

```
# import the nececessary lib
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```



## Data Augmentation for Training Variable

The statement `train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.2, horizontal_flip = True)`, as explained earlier, creates an instance of the `ImageDataGenerator` class and applies data augmentation techniques to the training data. This involves generating modified versions of the training images, such as rescaling, zooming, and flipping, to provide the model with a wider range of examples to learn from. This helps to prevent overfitting and improve the model's generalization ability.

Command for it is

```
# data augmentation for the training variable

train_datagen = ImageDataGenerator(rescale
=1./255, zoom_range=0.2, horizontal_flip = True)
```

## Data Augmentation for Testing Variable

The statement `test_datagen = ImageDataGenerator(rescale =1./255)` creates another instance of the `ImageDataGenerator` class, but with different augmentation parameters. This `test_datagen` object will be used to generate augmented versions of the testing data. The `rescale` parameter ensures that the pixel values of the testing images are also rescaled to the same range as the training data, which is important for consistent evaluation.

Command for it is

```
# data augmentation for the testing variable

test_datagen = ImageDataGenerator(rescale =1./255)
```

## data augmentation on the training data

This statement generates batches of augmented training data from a directory of images. It uses the `ImageDataGenerator` object `train_datagen` to apply data augmentation techniques, such as rescaling, zooming, and flipping, to the images. The augmented images are then resized to 64x64 pixels and loaded into batches of 100 images. The `class_mode` argument specifies that the images should be classified into multiple categories, in this case, COVID-19 or non-COVID-19.

In summary, this statement prepares the training data for an image classification task, specifically COVID-19 detection, by augmenting and preprocessing the images

Command for it is

```
# data augmentation on the training data

x_train =
train_datagen.flow_from_directory('/content/drive/MyDrive/covid19_prediction
/Data/train',
                                target_size=(64,64),
                                class_mode = 'categorical',
                                batch_size = 100)
```

### **data augmentation on the testing data**

This statement utilizes the test\_datagen object, initialized earlier for data augmentation, to generate batches of augmented test data. It specifies the directory containing the test images (/content/drive/MyDrive/covid19\_prediction /Data/test), resizes the images to 64x64 pixels (target\_size=(64,64)), maintains categorical class labels (class\_mode = 'categorical'), and processes the data in batches of 100 images (batch\_size = 100). The resulting x\_test variable holds the augmented test data batches.

Command for it is

```
# data augmentation on the testing data

x_test =
test_datagen.flow_from_directory('/content/drive/MyDrive/covid19_prediction
/Data/test',
                                target_size=(64,64),
                                class_mode = 'categorical',
                                batch_size = 100)
```

```
▼ Data Augmentation

[4] # import the necessary lib
    from tensorflow.keras.preprocessing.image import ImageDataGenerator

# data augmentation for the training variable
train_datagen = ImageDataGenerator(rescale =1./255, zoom_range=0.2, horizontal_flip = True)

[6] # data augmentation for the testing variable
test_datagen = ImageDataGenerator(rescale =1./255)

# data augmentation on the training data
x_train = train_datagen.flow_from_directory('/content/drive/MyDrive/covid19_prediction /Data/train',
                                             target_size=(64,64),
                                             class_mode = 'categorical',
                                             batch_size = 100)

Found 5144 images belonging to 3 classes.

[8] # data augmentation on the testing data
x_test = test_datagen.flow_from_directory('/content/drive/MyDrive/covid19_prediction /Data/test',
                                           target_size=(64,64),
                                           class_mode = 'categorical',
                                           batch_size = 100)

Found 1288 images belonging to 3 classes.
```

## ACTIVITY 3 : CNN Model building

Implement the necessary libraries for cnn deep learning algorithm

These statements are importing modules and classes from the TensorFlow Keras library for building convolutional neural networks (CNNs).

from tensorflow.keras.models import Sequential:

This statement imports the Sequential class from the tensorflow.keras.models module. The Sequential class is used to define and build CNN models by stacking layers sequentially.

from tensorflow.keras.layers import Convolution2D, MaxPooling2D, Flatten, Dense:

This statement imports several classes from the tensorflow.keras.layers module:

**Convolution2D:** This class represents a convolutional layer, which is a core building block of CNNs. It performs convolutions to extract features from input images.

**MaxPooling2D:** This class represents a max-pooling layer, which performs downsampling by taking the maximum value from a rectangular region of the input feature maps.

**Flatten:** This class represents a flattening layer, which converts the output of a convolutional layer into a one-dimensional vector suitable for fully connected layers.

**Dense:** This class represents a fully connected layer, which performs linear transformations on the input data. It is commonly used in the final stages of CNNs for classification or regression tasks.

command for it is :

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D, MaxPooling2D, Flatten, Dense
```

### adding layers

```
model = Sequential()

model.add(Convolution2D(32, (3,3), activation='relu', input_shape=(64,64,3))) #convolution layer
model.add(MaxPooling2D(pool_size=(2,2))) # maxpooling layer
model.add(Flatten()) # flatten layer

model.add(Dense(300, activation='relu')) # hidden layer 1
model.add(Dense(150, activation='relu')) # hidden layer 2

model.add(Dense(3, activation='softmax')) # output layer
```

### compile the model

The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process. Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

```
# compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

### ▼ CNN Model building

```
[ ] from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Convolution2D, MaxPooling2D, Flatten, Dense

[ ] # adding layers

    model = Sequential()

    model.add(Convolution2D(32, (3,3), activation='relu', input_shape=(64,64,3))) #convolution layer
    model.add(MaxPooling2D(pool_size=(2,2))) # maxpooling layer
    model.add(Flatten()) # flatten layer

    model.add(Dense(300, activation='relu')) # hidden layer 1
    model.add(Dense(150, activation='relu')) # hidden layer 2

    model.add(Dense(3, activation='softmax')) # output layer

[ ] # compile the model
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

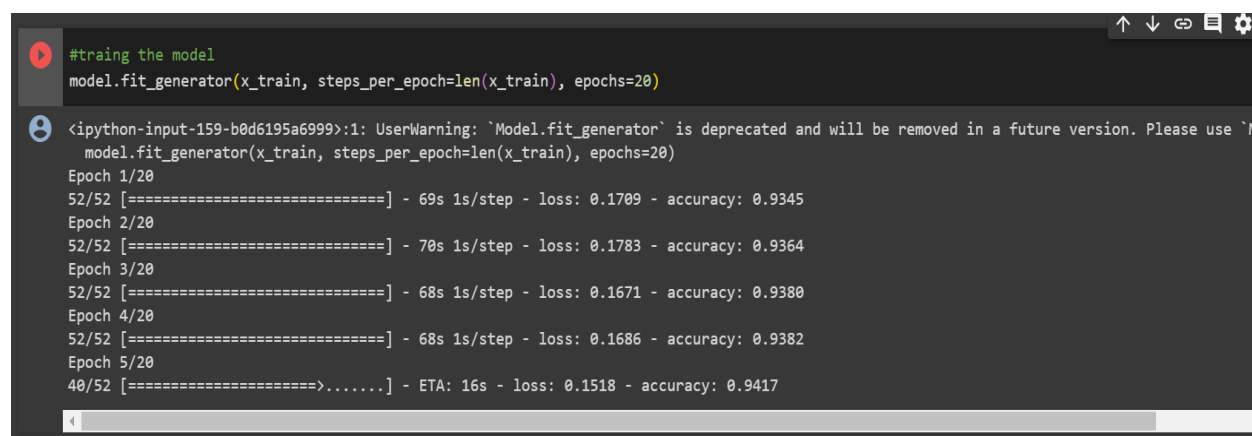
## activity 4 : training the model.

Let's commence the training of our model using the image dataset. The training process will span 25 epochs, and after each epoch, the model's state will be saved if it exhibits the lowest loss encountered thus far. We can observe that the training loss diminishes progressively until around the 10th epoch, suggesting potential for further model refinement.

Explanation of 'fit\_generator' function:

The fit\_generator function is employed to train a deep learning neural network. It takes the following arguments:

- steps\_per\_epoch: This parameter defines the total number of iterations to be drawn from the generator upon completion of an epoch and the commencement of the next. The steps\_per\_epoch value can be determined by dividing the total number of samples in the dataset by the batch size
- Epochs: This integer parameter specifies the desired number of training epochs for the model.



```
#training the model
model.fit_generator(x_train, steps_per_epoch=len(x_train), epochs=20)

<ipython-input-159-b0d6195a6999>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit` instead.
model.fit_generator(x_train, steps_per_epoch=len(x_train), epochs=20)
Epoch 1/20
52/52 [=====] - 69s 1s/step - loss: 0.1709 - accuracy: 0.9345
Epoch 2/20
52/52 [=====] - 70s 1s/step - loss: 0.1783 - accuracy: 0.9364
Epoch 3/20
52/52 [=====] - 68s 1s/step - loss: 0.1671 - accuracy: 0.9380
Epoch 4/20
52/52 [=====] - 68s 1s/step - loss: 0.1686 - accuracy: 0.9382
Epoch 5/20
40/52 [=====>.....] - ETA: 16s - loss: 0.1518 - accuracy: 0.9417
```

## Activity 5 : save the model

The model is saved with .h5 extension as follows An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data

```
# saving the model

model.save('covid19.h5')
```


## Activity 6 : Testing the Model Model

testing is the process of evaluating the performance of a deep learning model on a dataset that it has not seen before. It is a crucial step in the development of any machine learning model, as it helps to determine how well the model can generalize to new data and also import necessary libraries

```
from tensorflow.keras.preprocessing import image
import numpy as np

[ ] # testing 1
img = image.load_img('/content/drive/MyDrive/covid19_prediction /Data/test/COVID19/COVID19(573).jpg',target_size =(64,64))

[ ] img



[ ] x = image.img_to_array(img)
x = np.expand_dims(x,axis = 0)
pred =np.argmax(model.predict(x))
op =['COVID19','NORMAL','PNEUMONIA']
op[pred]

1/1 [=====] - 0s 17ms/step
'PNEUMONIA'
```

We are giving covid19 image but it is predicting the pneumonia so we are getting wrong prediction which it effects the humans health so we are going with different algorithm transfer learning are greater accuracy for predicting covid19

## Evaluating cnn model

```
evaluating a cnn model

model.evaluate(x_test)

13/13 [=====] - 18s 1s/step - loss: 0.2029 - accuracy: 0.9278
[0.20286519825458527, 0.9277950525283813]
```

And we are getting 92% accuracy for our cnn model but for more accuracy greater than 92% now we are going with transfer learning

## Activity 7 : creating testing and training datasets for vgg16

To build a DL model we have to split training and testing data into two separate folders. But In the project dataset folder training and testing folders are presented. So, in this case we just have to assign a variable and pass the folder path to it. Four different transfer learning models are used in our project and the best model (vgg16) is selected. The image input size of vgg16 model is 224,224.

```
trainpath = '/content/drive/MyDrive/covid19_prediction /Data/train'
testpath = '/content/drive/MyDrive/covid19_prediction /Data/test'
```

## Configure ImageDataGenerator class

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation. There are five main types of data augmentation techniques for image data; specifically:

Image shifts via the `width_shift_range` and `height_shift_range` arguments.

The image flips via the `horizontal_flip` and `vertical_flip` arguments.

Image rotations via the `rotation_range` argument.

Image brightness via the `brightness_range` argument.

Image zoom via the `zoom_range` argument.

An instance of the ImageDataGenerator class can be constructed for train and test.

```
train_datagen = ImageDataGenerator(rescale = 1./255, zoom_range=
0.2, shear_range= 0.2)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

## Apply ImageDataGenerator functionality to Train set and Test set

```
train = train_datagen.flow_from_directory(trainpath, target_size
=(224,224), batch_size = 16)
test = test_datagen.flow_from_directory(testpath, target_size
=(224,224), batch_size = 16)
```

## activity 8 : implementing vgg16

### implementing transfer learning using vgg16

```
[ ] trainpath = '/content/drive/MyDrive/covid19_prediction/Data/train'
    testpath = '/content/drive/MyDrive/covid19_prediction/Data/test'
```

```
[ ] train_datagen = ImageDataGenerator(rescale = 1./255, zoom_range= 0.2, shear_range= 0.2)
    test_datagen = ImageDataGenerator(rescale = 1./255)
```

```
▶ train = train_datagen.flow_from_directory(trainpath, target_size = (224, 224), batch_size = 16)
    test = test_datagen.flow_from_directory(testpath, target_size = (224, 224), batch_size = 16)
```

Found 5144 images belonging to 3 classes.  
Found 1288 images belonging to 3 classes.

```
[ ] from tensorflow.keras.applications.vgg16 import VGG16
    from tensorflow.keras.layers import Dense, Flatten
    from tensorflow.keras.models import Model
```

```
▶ vgg = VGG16(include_top = False, input_shape = (224, 224, 3))
```

```
[ ] for layer in vgg.layers:
    print(layer)
```

```
<keras.src.engine.input_layer.InputLayer object at 0x7805b2e716f0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x780494329960>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7805b2e735e0>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7805301dbd30>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x78054cbec8e0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x78049432a710>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7805b2fbd870>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7805b2e73a30>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7805b2fbc770>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x78049432a950>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7805b2fbc490>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7805b2fbc6e0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7805b2fbecb0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x78049432a4a0>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7804943396c0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7805b2fbff70>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7805b2fbec60>
```



```
[ ] len(vgg16.layers)

21

[ ] for layer in vgg.layers:
    layer.trainable = False

[ ] x= Flatten()(vgg.output)

[ ] output = Dense(3, activation = 'softmax')(x)

[ ] vgg16 = Model(vgg.input,output)
```

```
] vgg16.summary()
```

Model: "model\_7"

Layer (type)	Output Shape	Param #
=====		
input_11 (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_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_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808

```
block_7 (Block) (None, 14, 14, 512) 2359808

=====
Total params: 14789955 (56.42 MB)
Trainable params: 75267 (294.01 KB)
Non-trainable params: 14714688 (56.13 MB)
```

## Activity 9 : training vgg16 model and compile and saving it

```
[ ] vgg16.compile(loss = 'categorical_crossentropy',optimizer = 'adam',metrics =['accuracy'])

[ ] vgg16.fit(train,validation_data=test,epochs=2,steps_per_epoch=len(train),validation_steps =len(test))

Epoch 1/2
322/322 [=====] - 180s 549ms/step - loss: 0.2772 - accuracy: 0.9018 - val_loss: 0.2377 - val_accuracy: 0.9208
Epoch 2/2
322/322 [=====] - 158s 492ms/step - loss: 0.1446 - accuracy: 0.9493 - val_loss: 0.1509 - val_accuracy: 0.9433
<keras.src.callbacks.History at 0x780494340a90>

[ ] # saving the model

model.save('covid.h5')

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This API is deprecated. Please use `model.save(filepath, save_format='tf')` to save a model as a TensorFlow SavedModel instead.
```


And getting 95% accuracy which is are good predictions

## Activity 10 : testing vgg16 model

```
[ ] from tensorflow.keras.preprocessing import image
import numpy as np

[ ] # testing 1
img = image.load_img('/content/drive/MyDrive/covid19_prediction /Data/test/COVID19/COVID19(464).jpg',target_size =(64,64))

[ ] img



[ ] x = image.img_to_array(img)
x = np.expand_dims(x,axis = 0)
pred =np.argmax(model.predict(x))
op =['COVID19','NORMAL','PNEUMONIA']
op[pred]

1/1 [=====] - 0s 158ms/step
'COVID19'
```

Therefore we are getting correct and greater accurate predictions by vgg16

## Activity 11 : Application Building In this section

we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

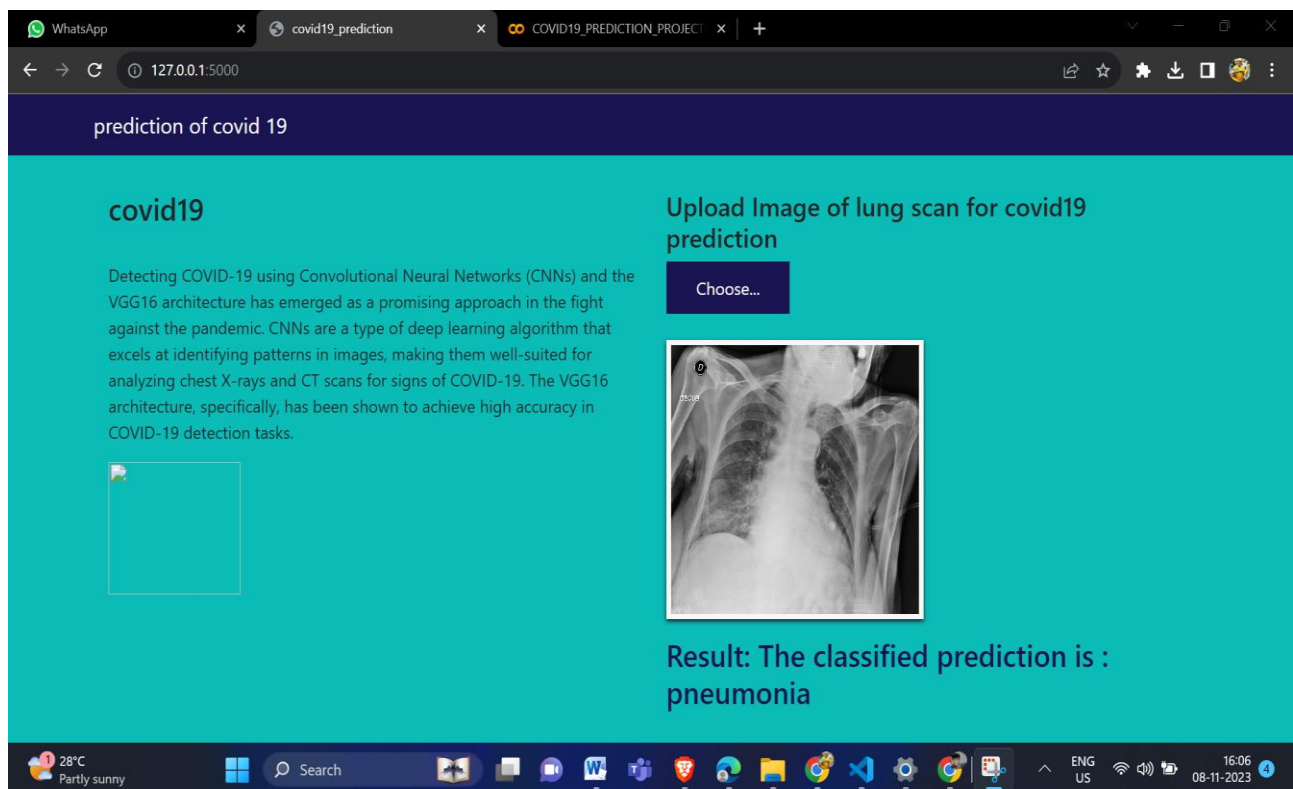
This section has the following tasks

- Building HTML Pages
- Building server side script

Building Html Pages: For this project create one HTML file namely

- Index.html

Let's see how our index.html page looks like:



## Activity 12: Build Python code

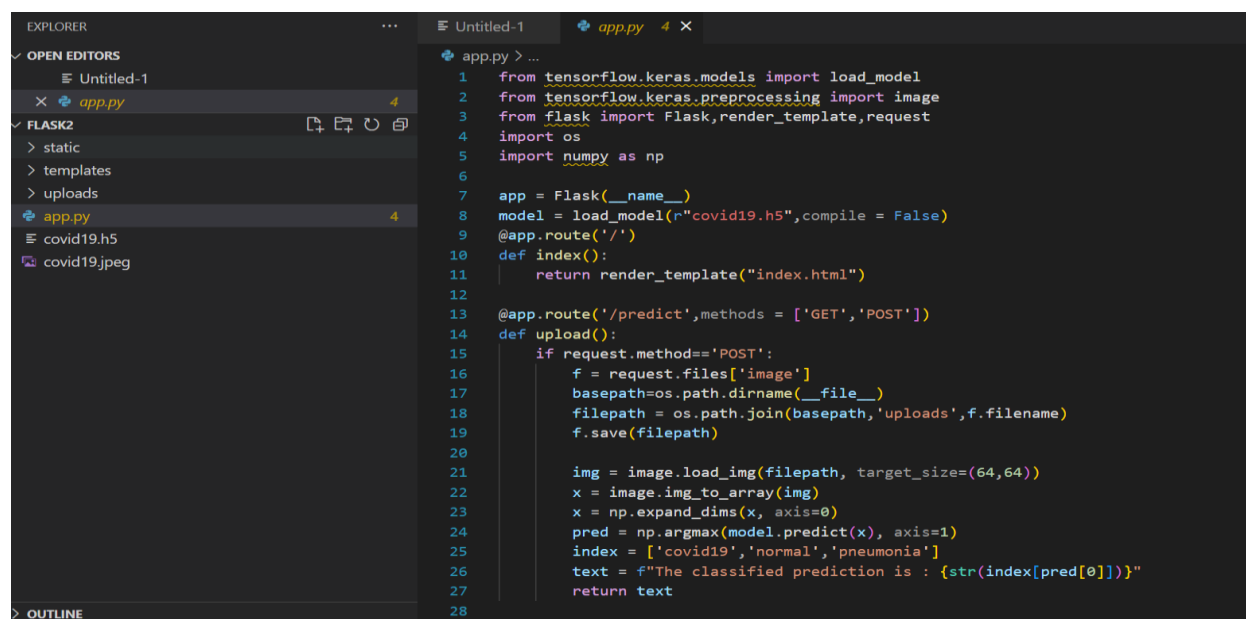
Import the libraries

Loading the saved model and initializing the flask app

Render HTML pages:

Once we uploaded the file into the app, then verifying the file uploaded properly or not. Here we will be using declared constructor to route to the HTML page which we have created earlier. In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method

Here we are routing our app to res function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the index.html page earlier.



```
1 from tensorflow.keras.models import load_model
2 from tensorflow.keras.preprocessing import image
3 from flask import Flask, render_template, request
4 import os
5 import numpy as np
6
7 app = Flask(__name__)
8 model = load_model(r"covid19.h5", compile = False)
9 @app.route('/')
10 def index():
11     return render_template("index.html")
12
13 @app.route('/predict', methods = ['GET', 'POST'])
14 def upload():
15     if request.method == 'POST':
16         f = request.files['image']
17         basepath = os.path.dirname(__file__)
18         filepath = os.path.join(basepath, 'uploads', f.filename)
19         f.save(filepath)
20
21         img = image.load_img(filepath, target_size=(64,64))
22         x = image.img_to_array(img)
23         x = np.expand_dims(x, axis=0)
24         pred = np.argmax(model.predict(x), axis=1)
25         index = ['covid19', 'normal', 'pneumonia']
26         text = f"The classified prediction is : {str(index[pred[0]])}"
27         return text
28
```

## Main function

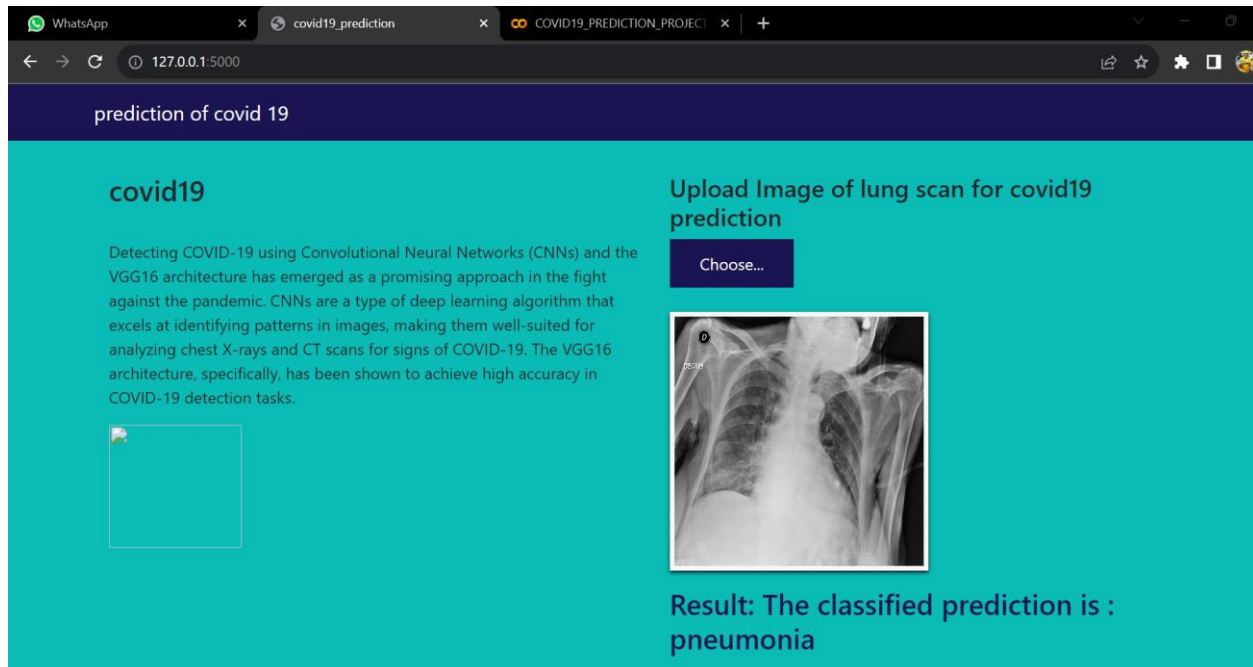
```
if __name__ == '__main__':  
    app.run(debug=True)
```

### Activity 3: Run the application

- Open the Anaconda prompt from the start menu.
- Navigate to the folder where your Python script is.
- Now type the “python app.py” command.
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
base) C:\Users\91984\Documents\covid19 project\flask2>python app.py  
023-11-08 16:45:15.896680: I tensorflow/core/platform/cpu_feature_guard.cc:193] Using TensorFlow with available CPU instructions in performance-critical operations.  
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2, the compiler flags must be passed with the appropriate compiler flags.  
* Serving Flask app 'app'  
* Debug mode: on  
WARNING: This is a development server. Do not use it in a production deployment.  
* Running on http://127.0.0.1:5000  
Press CTRL+C to quit  
* Restarting with watchdog (windowsapi)
```

The home page looks like this. When you click on the button “Drop in the image you want to validate!”, you’ll be redirected to the predict section



If we click the predict button it will shows the ouyput as pneumonia (1<sup>st</sup> stage of covid19)

-----The end-----







