# Covid-19 Detection from Lung X-rays

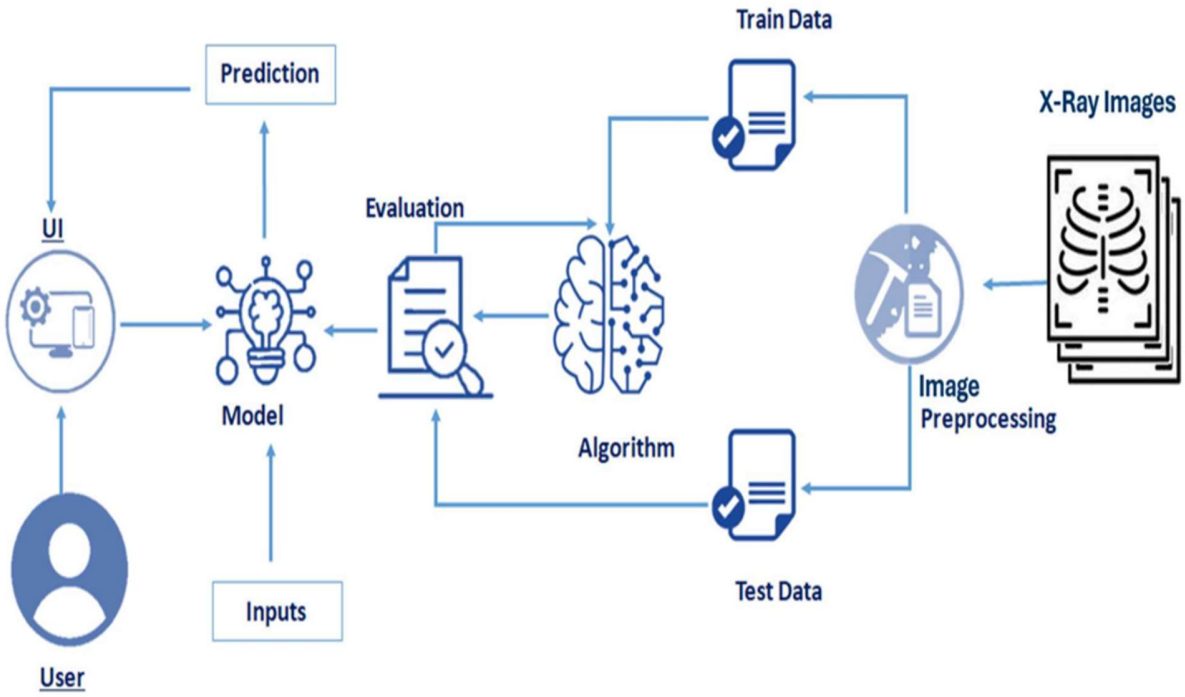
**Project Description:**

COVID-19 (coronavirus disease 2019) is an infectious disease caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2), which is a strain of coronavirus. The disease was officially announced as a pandemic by the World Health Organization(WHO) on 11 March 2020. Given spikes in new COVID-19 cases and the re-opening of daily activities around the world, the demand for curbing the pandemic is to be more emphasized. Medical images and artificial intelligence (AI) have been found useful for rapid assessment to provide treatment of COVID-19 infected patients. The PCR test may take several hours to become available, information revealed from the chest X-ray plays an important role for a rapid clinical assessment. This means if the clinical condition and the chest X-ray are normal, the patient is sent home while awaiting the results of the etiological test. But if the X-ray shows pathological findings, the suspected patient will be admitted to the hospital for close monitoring. Chest X-ray data have been found to be very promising for assessing COVID-19 patients, especially for resolving emergency departments and urgent-care-center overcapacity. Deep-learning (DL) methods in artificial intelligence (AI) play a dominant role as high-performance classifiers in the detection of disease using chest X-rays.

One of the biggest challenges following the Covid-19 pandemic is the detection of the disease in patients. To address this challenge we have been using the Deep Learning Algorithm to build an image recognition model that can detect the presence of COVID-19 from an X-ray or CT-Scanimage of a patient's lungs.

Transfer learning has become one of the most common techniques that has achieved better performance in many areas, especially in medical image analysis and classification. We used Transfer Learning techniques like Inception V3, Resnet50, Xception V3, and stacked model which uses both MobileNet and Resnet 50 which are more widely used as transfer learning methods in medical image analysis and are highly effective.

**Technical Architecture:**



## Prerequisites:

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

- **Anaconda navigator and PyCharm / Spyder:**
  - Refer to the link below to download Anaconda Navigator
  - Link (PyCharm) : https://youtu.be/1ra4zH2G4o0
  - Link (Spyder) : https://youtu.be/5mDYijMfSzs
- **Python packages:**
  - Open anaconda prompt as administrator
  - Type "pip install numpy" and click enter.
  - Type "pip install pandas" and click enter..
  - Type "pip install tensorflow.
  - Type "pip install keras 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
  - **ResNet-50:** https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
  - **Inception-V3:** https://iq.opengenus.org/inception-v3-model-architecture/
  - **Xception:** https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/
- **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=lj4I_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.

**Project Flow:**

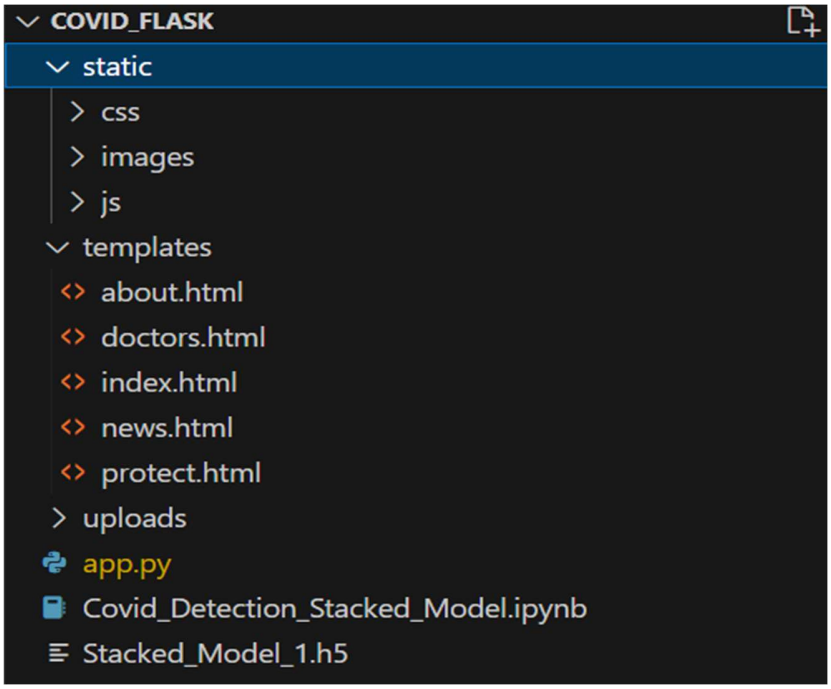
- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analyzed by the model which is integrated with flask application.
- The Stacked Model analyzes the image, then the prediction is showcased on the Flask UI.

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

- Data Collection.
    - Create a Train and Test path.
- Data Pre-processing.
    - Import the required library
    - Configure Image DataGenerator class
    - Apply Image DataGenerator functionality to Trainset and Testset
- 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
- Application Building
    - Create an HTML file
    - Build Python Code

**Project Structure:**

Create a Project folder which contains files as shown below




- Flask folder consists of static, templates, pynb notebook and app.py
- Training file consist of Covid_Detection_Stacked_Model.ipynb which has the model definition and it's training
- Requirements.txt file consists of libraries and modules required for building this project.

## Milestone 1: Data Collection

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

**Activity 1: Download the dataset**

Collect images of COVID-19 chest X-ray images then organize 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 4 types of chest X-ray images like COVID-19, Lung Opacity, Pneumonia and normal. They are saved in the respective sub-directories with their respectivenames.

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

Dataset:-[https://www.kaggle.com/code/rollanmaratov/covid19-detection-using-tensorflow-from-chest-xray/data](https://www.kaggle.com/code/rollanmaratov/covid19-detection-using-tensorflow-from-chest-xray/data)

**Note: For better accuracy train on more images**

We are going to build our training model on Google colab and Kaggle.

Since the dataset which are using is from Kaggle we created an temp directory to hold the images of all four sub classes

```
!mkdir "/tmp/X-RAY/"
```

Python

```
!mkdir -p tmp/X-RAY/COVID
!mkdir -p tmp/X-RAY/LUNG_OPACITY
!mkdir -p tmp/X-RAY/NORMAL
!mkdir -p tmp/X-RAY/PNEUMONIA
```

Python

```
!cp /kaggle/input/covid19-radiography-database/COVID-19_Radiography_Dataset/COVID/images/*.png tmp/X-RAY/COVID/
!cp /kaggle/input/covid19-radiography-database/COVID-19_Radiography_Dataset/Normal/images/*.png tmp/X-RAY/NORMAL/
!cp /kaggle/input/covid19-radiography-database/COVID-19_Radiography_Dataset/Lung_Opacity/images/*.png tmp/X-RAY/LUNG_OPACITY/
!cp /kaggle/input/covid19-radiography-database/COVID-19_Radiography_Dataset/Viral\ Pneumonia/images/*.png tmp/X-RAY/PNEUMONIA
```

## Activity 2: Create training and testing dataset

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 is selected. The image input size of the selected model is 224, 224.

```
train=train_datagen.flow_from_directory(train_path,target_size=(224,224),batch_size=16,subset='training',class_mode='categoric
test=train_datagen.flow_from_directory(train_path,target_size=(224,224),batch_size=16,subset='validation',class_mode='categori
```

Python

## Milestone 2: Image Preprocessing

In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Link : https://thesmartbridge.com/documents/spsaimldocs/CNNprep.pdf

### Activity 1: Importing the libraries

Import the necessary libraries as shown in the image

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os

import keras
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report,accuracy_score
from keras.layers import concatenate
from keras.layers import Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import MaxPooling2D, Flatten,Conv2D, Dense,BatchNormalization,GlobalAveragePooling2D,Dropout
from tensorflow.keras.applications.densenet import DenseNet169
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2
```

### Activity 2: 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.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_path="tmp/X-RAY/"


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

**Activity 3: Apply Image Data Generator functionality to Train set and Test set**

Let us apply Image Data Generator functionality to the Train set and Test set by using the following code. For Training set using flow_from_directory function.

This function will return batches of images from the subdirectories

Arguments:

- directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
- batch_size: Size of the batches of data which is 16.
- target_size: Size to resize images after they are read from disk.
- class_mode:
  - ‘int': means that the labels are encoded as integers (e.g. for sparse categorical_crossentropy loss).
  - 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).
  - 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).
  - None (no labels).

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

Python

```
train=train_datagen.flow_from_directory(train_path,target_size=(224,224),batch_size=16,subset='training',class_mode='categoric
test=train_datagen.flow_from_directory(train_path,target_size=(224,224),batch_size=16,subset='validation',class_mode='categori
```

Python

```
Found 16933 images belonging to 4 classes.
Found 4232 images belonging to 4 classes.
```

Total the dataset is having 16933train images and 4232 test images divided under 4 classes

## Milestone 3: Model Building

Now it's time to build our model. Let’s use the pre-trained model which is a combination MobileNet and Resnet50 , one of the convolution neural net (CNN) architecture which is considered as a very good model for Image classification. Also we tried with different transfer learning model for this application and based on the validation accuracy the final model was chosen

| Model | Training Accuracy | Validation Accuracy |
|---|---|---|
| VGG-16 | **90.37** | 89.27 |
| Resnet 50 | 66.78 | 64.89 |
| Inception | 90.24 | 88.5 |
| Exception | 88.04 | 84.90 |
| Efficient Net | 48.05 | 48.16 |
| Stacked Model (Resnet, Mobile_Net) | 88.90 | **90.17** |

**Activity 1: Pre-trained CNN model as a Feature Extractor**

For one of the models, we will use it as a simple feature extractor by freezing all the five convolution blocks to make sure their weights don't get updated after each epoch as we train our own model.

Here, we have considered images of dimension (224,224,3).

Also, we have assigned include_top = False because we are using convolution layer for features extraction and wants to train a fully connected layer for our images classification(since it is not the part of Imagenet dataset)

Flatten layer flattens the input. Does not affect the batch size.

```
input_shape = (224,224,3)
input_layer = Input(shape = (224, 224, 3))
mobilenet_base = MobileNetV2(weights = 'imagenet',input_shape = input_shape,include_top = False)
densenet_base = DenseNet169(weights = 'imagenet', input_shape = input_shape,include_top = False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilene
9406464/9406464 [==============================] - 1s 0us/step
```

```
for layer in mobilenet_base.layers:
    layer.trainable =  False
for layer in densenet_base.layers:
    layer.trainable = False
```

**Activity 2: Adding Dense Layers**

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer. Let us create a model object named model with inputs as resnet.input, mobilenet.input and output as combination of dense layers as mentioned below.

```
model_densenet = densenet_base(input_layer)
model_densenet = GlobalAveragePooling2D()(model_densenet)
output_densenet = Flatten()(model_densenet)

merged = tf.keras.layers.Concatenate()([output_mobilenet, output_densenet])

x = BatchNormalization()(merged)
x = Dense(256,activation = 'relu')(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(128,activation = 'relu')(x)
x = Dropout(0.5)(x)
x = Dense(4, activation = 'softmax')(x)
stacked_model = tf.keras.models.Model(inputs = input_layer, outputs = x)
```

The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities. Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

```
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         []

 mobilenetv2_1.00_224 (Func  (None, 7, 7, 1280)        2257984   ['input_1[0][0]']
 tional)

 densenet169 (Functional)    (None, 7, 7, 1664)        1264288   ['input_1[0][0]']
                                                       0

 global_average_pooling2d (  (None, 1280)              0         ['mobilenetv2_1.00_224[0][0]']
 GlobalAveragePooling2D)

 global_average_pooling2d_1  (None, 1664)              0         ['densenet169[0][0]']
  (GlobalAveragePooling2D)

 flatten (Flatten)           (None, 1280)              0         ['global_average_pooling2d[0][
                                                                  0]']

 flatten_1 (Flatten)         (None, 1664)              0         ['global_average_pooling2d_1[0
                                                                  ][0]']

 concatenate (Concatenate)   (None, 2944)              0         ['flatten[0][0]',
...
Total params: 15700996 (59.89 MB)
Trainable params: 793732 (3.03 MB)
Non-trainable params: 14907264 (56.87 MB)
```

**Activity 3: Configure the Learning Process**

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

```
optm = Adam(lr=0.0001)
stacked_model.compile(loss='categorical_crossentropy', optimizer=optm,
                      metrics=['accuracy'])
```

**Activity 4: Train the model**

Now, let us train our model with our image dataset. The model is trained for 20 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 20 epochs and probably there is further scope to improve the model.

**Arguments:**

- fit functions used to train a deep learning neural network
- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

- Epochs: an integer and number of epochs we want to train our model for.

- validation_data can be either:
  - an inputs and targets list
  - a generator
  - an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.

- validation_steps: only if the validation_data is a generator then only this argument can be used.

```python
from keras.callbacks import EarlyStopping
EarlyStopping = EarlyStopping(monitor='val_accuracy',
                              min_delta=.01,
                              patience=6,
                              verbose=1,
                              mode='auto',
                              baseline=None,
                              restore_best_weights=True)
```

```python
epochs = 20
batch_size  = 16
stacked_history = stacked_model.fit(train,
                        steps_per_epoch = 16933 // batch_size,
                        epochs = 20,
                        validation_data = test,
                        callbacks=[EarlyStopping])
```

```
Epoch 1/20
1058/1058 [==============================] - 249s 236ms/step - loss: 0.4361 - accuracy: 0.8395 - val_loss: 0.3077 - val_accuracy: 0.8922
Epoch 2/20
1058/1058 [==============================] - 251s 237ms/step - loss: 0.4205 - accuracy: 0.8438 - val_loss: 0.3229 - val_accuracy: 0.8837
Epoch 3/20
1058/1058 [==============================] - 249s 236ms/step - loss: 0.3966 - accuracy: 0.8528 - val_loss: 0.2901 - val_accuracy: 0.8956
Epoch 4/20
1058/1058 [==============================] - 253s 239ms/step - loss: 0.3929 - accuracy: 0.8562 - val_loss: 0.2808 - val_accuracy: 0.9000
Epoch 5/20
1058/1058 [==============================] - 251s 237ms/step - loss: 0.3720 - accuracy: 0.8663 - val_loss: 0.3046 - val_accuracy: 0.8963
Epoch 6/20
1058/1058 [==============================] - 249s 235ms/step - loss: 0.3756 - accuracy: 0.8613 - val_loss: 0.2865 - val_accuracy: 0.8993
Epoch 7/20
1058/1058 [==============================] - 248s 234ms/step - loss: 0.3676 - accuracy: 0.8663 - val_loss: 0.2687 - val_accuracy: 0.9067
Epoch 8/20
1058/1058 [==============================] - 252s 238ms/step - loss: 0.3562 - accuracy: 0.8711 - val_loss: 0.2726 - val_accuracy: 0.9060
Epoch 9/20
1058/1058 [==============================] - 250s 237ms/step - loss: 0.3587 - accuracy: 0.8724 - val_loss: 0.2813 - val_accuracy: 0.9003
Epoch 10/20
1058/1058 [==============================] - 252s 238ms/step - loss: 0.3440 - accuracy: 0.8772 - val_loss: 0.2735 - val_accuracy: 0.9081
Epoch 11/20
1058/1058 [==============================] - 252s 238ms/step - loss: 0.3395 - accuracy: 0.8765 - val_loss: 0.2656 - val_accuracy: 0.9041
Epoch 12/20
1058/1058 [==============================] - 251s 238ms/step - loss: 0.3380 - accuracy: 0.8789 - val_loss: 0.2549 - val_accuracy: 0.9121
```

From the above run time, we can easily observe that at 11$^{th}$ epoch the model is givingthe better accuracy, which is initialized through call back parameter

### Activity 5 : 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.

```
y_pred = stacked_model.predict(test_generator)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_generator.classes

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred_classes)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
67/67 [==============================] - 15s 218ms/step
Accuracy: 90.17%
```

- In the above code, we have tested the model with a image of x-rays, which is produced by the test data generator
- Model which us trained is working better with unknown data also which is clear from the validation results, so we will save the model

## Milestone 4: 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.

### Milestone 5: 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

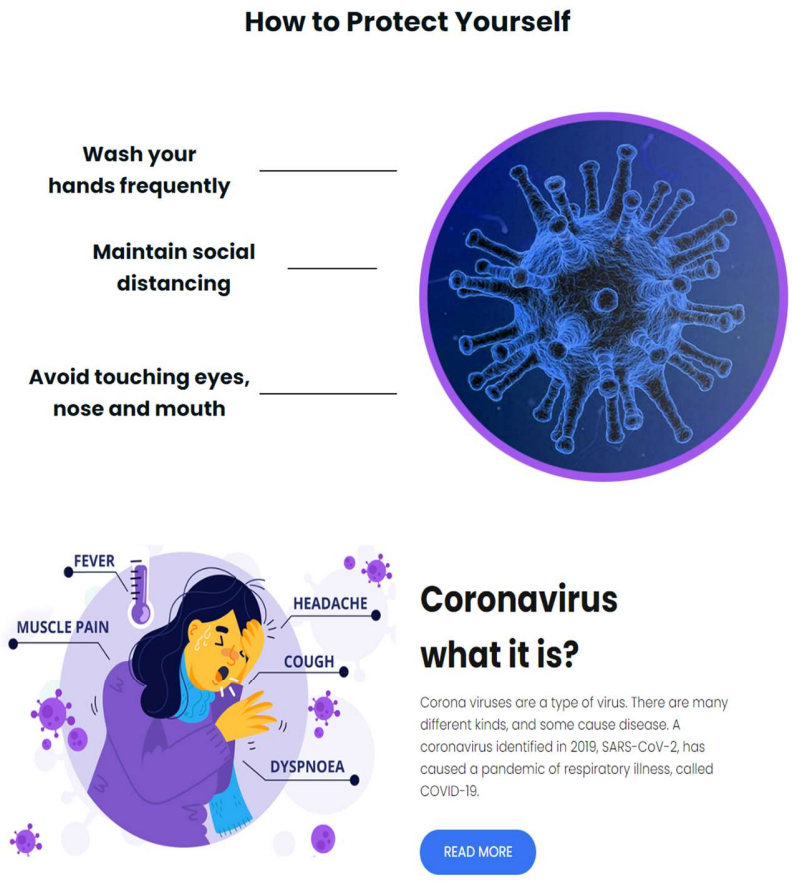
**Activity1: Building Html Pages:**

For this project create HTML files namely

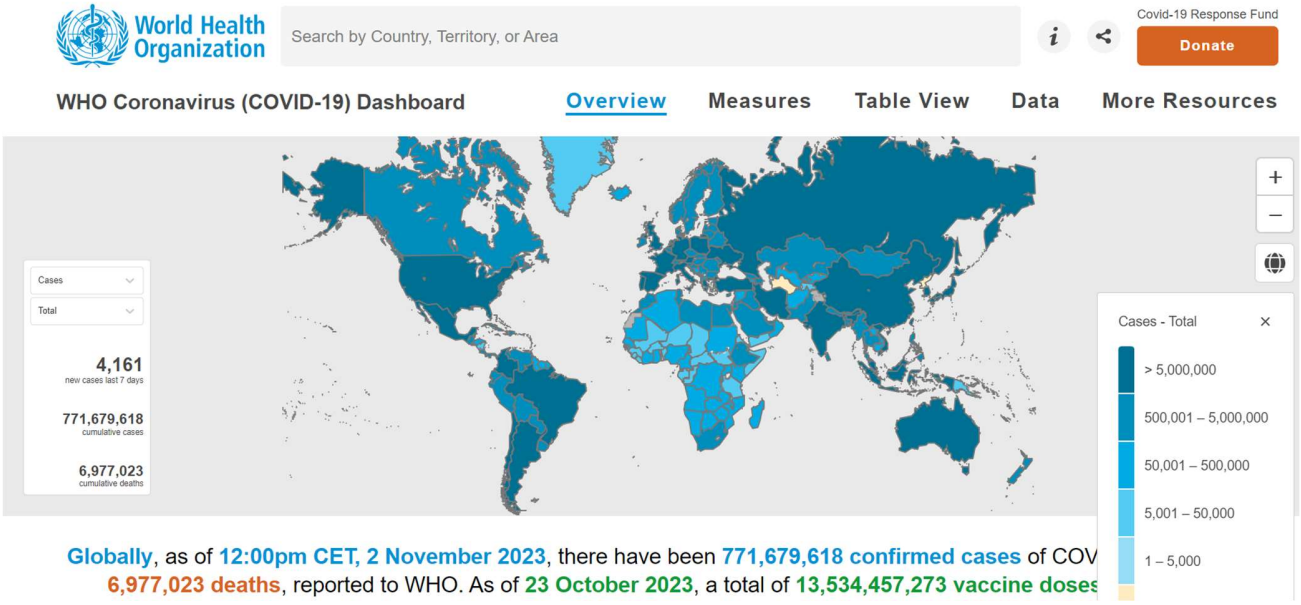
- index.html
- about.html
- doctors.html
- news.html
- protect.html

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

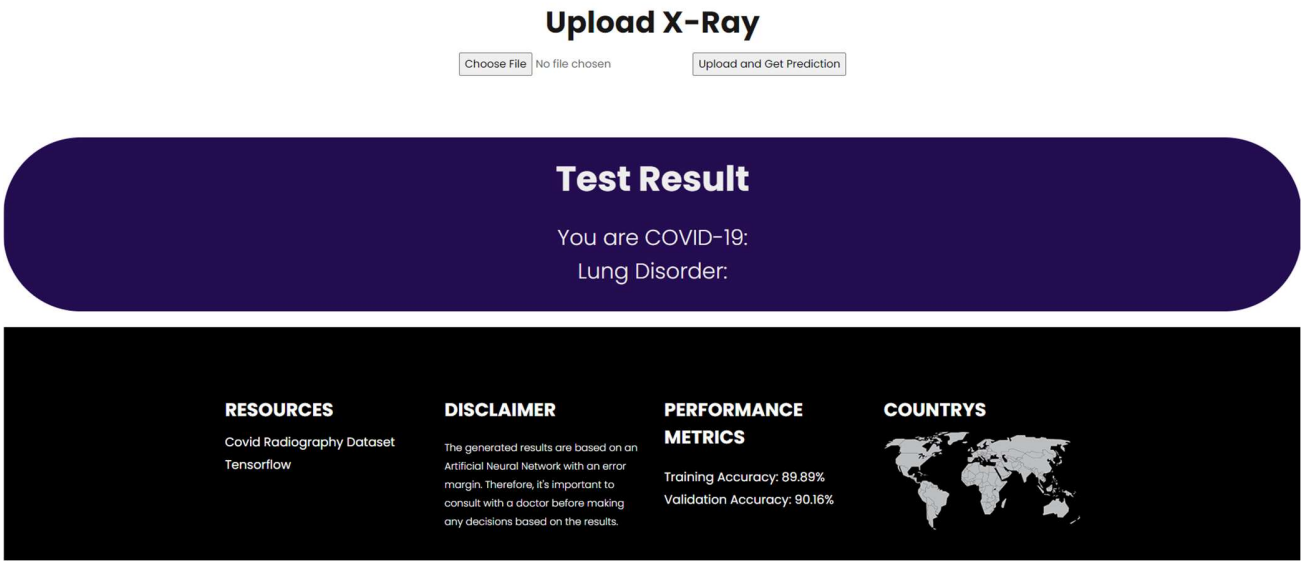

When you scroll down you will land on the following page



When you click on the read more button, it will redirect you to the below page




When you scroll further you will land on the actual X-ray based Covid prediction page, also the other related information is mentioned in the same page



**Activity 2: Build Python code:**

Import the libraries

```
from flask import Flask, request, render_template
from tensorflow.keras.preprocessing import image
import os
import numpy as np
from PIL import Image
import tensorflow as tf
from tensorflow import keras
```

Loading the saved model and initializing the flask app

```
def load_model():
    # Replace 'model_path' with the path to your trained model file.
    model_path = 'C:\\Users\\varun\\OneDrive\\Desktop\\Fall Sem\\Smart Internz AI&ML\\Covid Detection\\4._Project\\Covid_Flask\\Stacked_Model_1.h5'
    model = keras.models.load_model(model_path)
    return model

model = load_model()
```

Render HTML pages:

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/protect')
def protect():
    return render_template('protect.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/doctors')
def doctors():
    return render_template('doctors.html')

@app.route('/news')
def news():
    return render_template('news.html')
```

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.

```python
@app.route('/predict',methods = ['GET','POST'])
def upload():
    if request.method=='POST':
        f = request.files['image']
        basepath=os.path.dirname(__file__)
        filepath = os.path.join(basepath,'uploads',f.filename)
        f.save(filepath)

        img = image.load_img(filepath,target_size =(224,224))
        x = image.img_to_array(img)/255
        x = np.expand_dims(x,axis = 0)
        pred =np.argmax(model.predict(x),axis=1)
        print(int(pred))

        lis=['COVID','LUNG OPACITY','NORMAL','PNEUMONIA']
        # Process the prediction result and format it as needed
        if int(pred)==0:
            covid_prob="Positive"
        else:
            covid_prob="Negative"

        lung_disorder_prob = lis[int(pred)]

        return render_template('index.html', covid_prob=covid_prob, lung_disorder_prob=lung_disorder_prob)
```

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 rendered to the text that we have mentioned in the index.html page earlier.

**Main Function:**

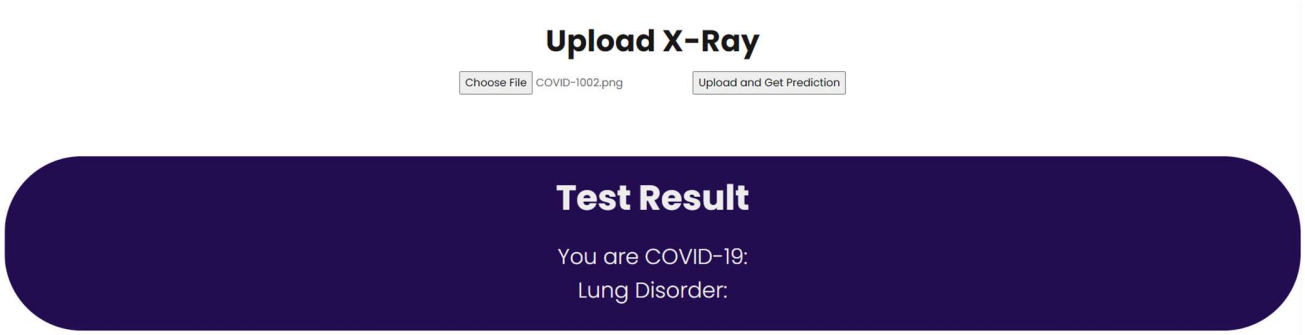
```python
if __name__ == '__main__':
    app.run(debug=True)
```

**Activity 3: Run the application**

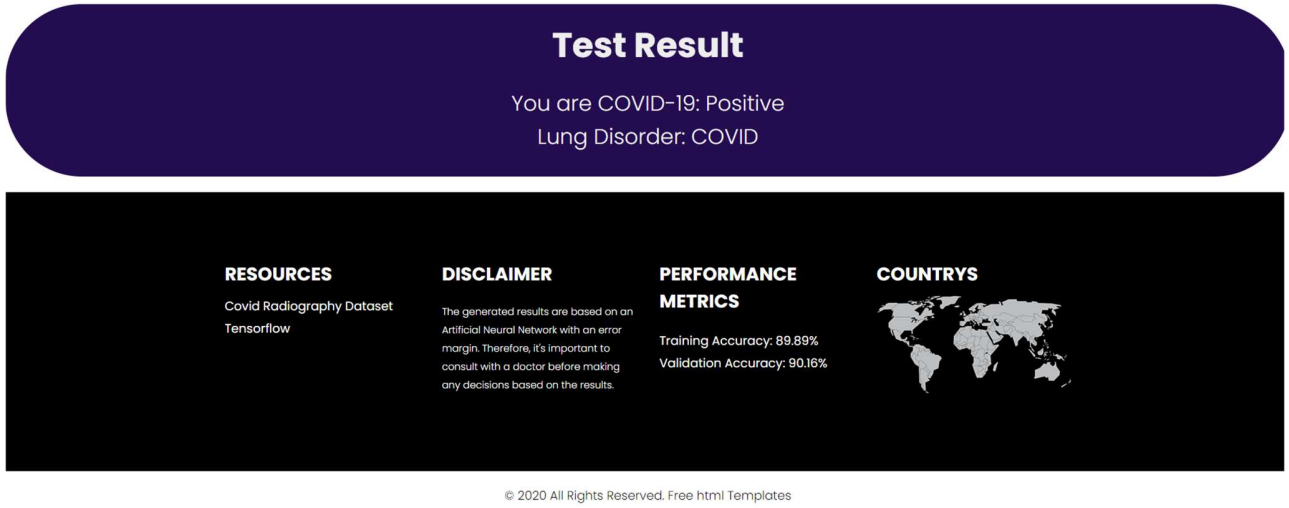
- Open the Anaconda prompt from the start menu.
- Navigate to the folder where your Python script is.
- Now type the “python app.py” command.
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button,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

```
Press CTRL+C to quit
 * Restarting with stat
2023-11-06 16:27:26.363028: I tensorflow/core/p
erations.
To enable the following instructions: SSE SSE2
 * Debugger is active!
 * Debugger PIN: 866-198-558
```

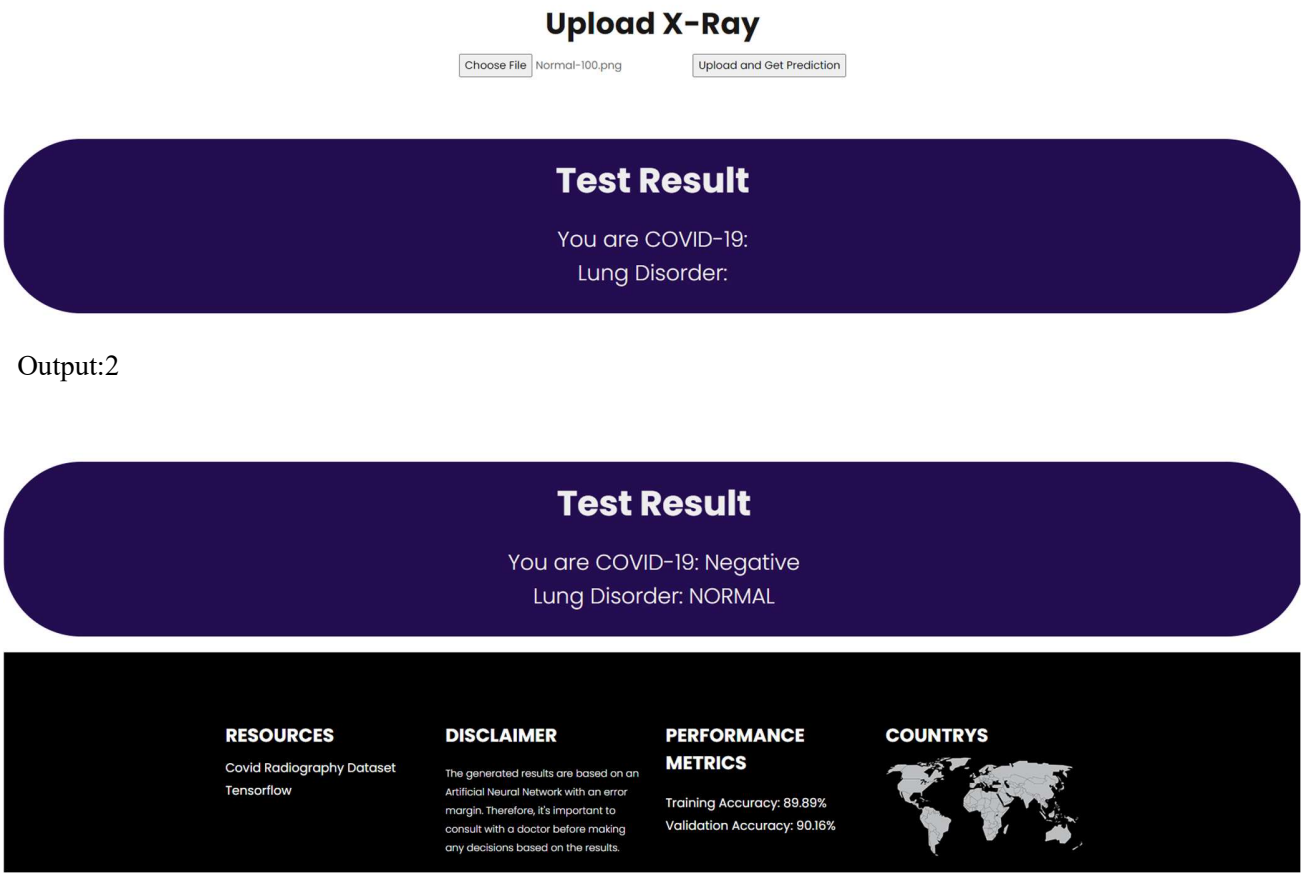
Input 1 (Covid)




Once you upload the image and click on Upload and get prediction button, the output will be displayed in the below page Output:1



Input:2(Normal)




Output:2