

Covid-19 Detection from Lung X-rays

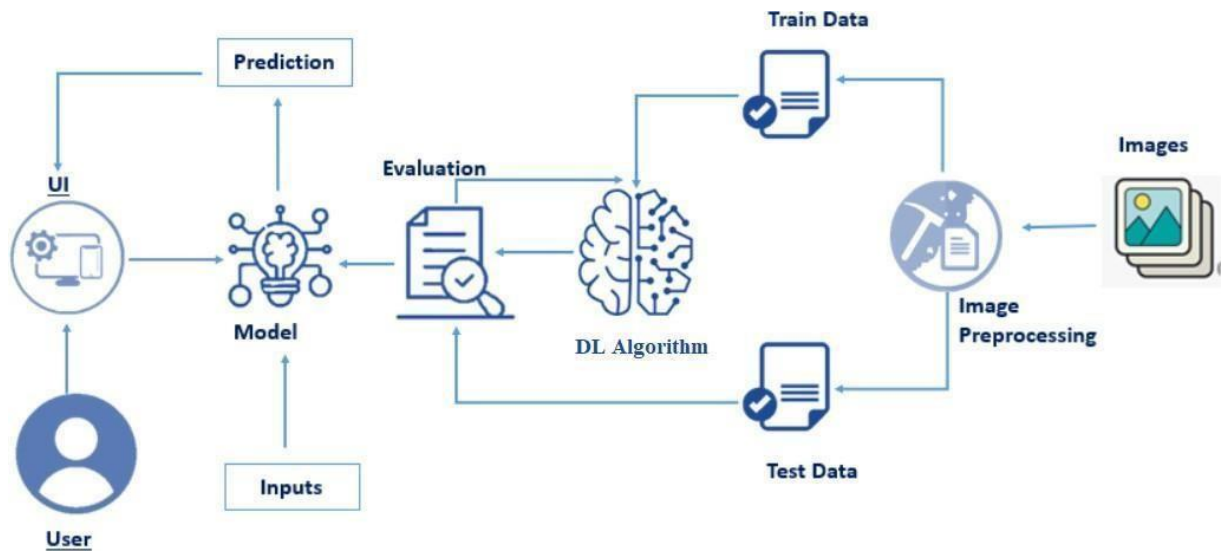
Project Description:

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-department 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 the 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-Scan image 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 that are more widely used as a transfer learning method in medical image analysis and they are highly effective.

Technical Architecture:



Prerequisites:

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

- **Anaconda navigator and PyCharm / Spyder:**

- Refer 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==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>
- **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 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. ○ Create a Train and Test path. ○ Data Pre-processing.
- Import the required library ○ Configure ImageDataGenerator class ○ Apply ImageDataGenerator 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

Name	Date modified	Type	Size
📁 covid_detection	16-11-2023 11:02 PM	File folder	
📁 static	18-11-2023 12:52 AM	File folder	
📁 templates	18-11-2023 01:37 AM	File folder	
📁 uploads	18-11-2023 02:07 AM	File folder	
📄 app	18-11-2023 02:16 AM	PY File	4 KB
📄 covid.h5	16-11-2023 11:51 PM	H5 File	23,063 KB

- Flask folder consists of static, templates and app.py
- covid_detection folder consists of train and test data
- Flask folder consist of trained model covid.h5

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 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 2 types of Covid-19 images like Covid-19 positive and Covid-19 negative 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/code/rollanmaratov/covid19-detection-using-tensorflow-fr-om-chest-xray/data>

Note: For better accuracy train on more images

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

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 and move the dataset to that folder and mount to that folder.

```
from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[ ] base_dir = os.path.join('/content/drive/MyDrive/', 'covid_detection')
    train_dir = os.path.join(base_dir, 'Train')
    validation_dir = os.path.join(base_dir, 'Test')
```

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 model is 500,500 .

```
base_dir = os.path.join('/content/drive/MyDrive/', 'covid_detection')
train_dir = os.path.join(base_dir, 'Train')
validation_dir = os.path.join(base_dir, 'Test')
```

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 tensorflow as tf
import os
import numpy as np
import pandas as pd
import keras
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

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.

```

image_gen_val = ImageDataGenerator(rescale=1./255)

val_data_gen = image_gen_val.flow_from_directory(batch_size=Batch_Size,
                                                  directory=validation_dir,
                                                  color_mode = 'grayscale',
                                                  shuffle=False,
                                                  target_size=(Image_Size,
Image_Size),
                                                  class_mode='binary')

```

Activity 3: Apply ImageDataGenerator functionality to Train set and Test set

Let us apply ImageDataGenerator 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 32.
- 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).


```
image_gen_train = ImageDataGenerator(  
    rescale=1./255,  
    shear_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True,  
)  
  
train_data_gen = image_gen_train.flow_from_directory(batch_size=Batch_Size,  
                                                    directory=train_dir,  
                                                    color_mode = 'grayscale',  
                                                    target_size=(Image_Size,Image_Size),  
                                                    class_mode='binary')
```

Found 453 images belonging to 2 classes.

Total the dataset is having 453 train images and 219 test images divided under 2 classes

Milestone 3: Model Building

Now it's time to build our model. Let's use the pre-trained model which is, one of the convolution neural net (CNN) architecture which is considered as a very good model for Image classification.

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 (500,500,1).

Also, we have assigned `include_top = False` because we are using convolution layer for features extraction and wants to train 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.

```

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(500, 500, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')
])

```

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 input and output as dense layer.

```

tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dense(256, activation='relu'),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.Dense(2, activation='softmax')
])

```

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.



Model: "sequential_1"

Layer (type)	Output Shape	Param #
=====		
conv2d_6 (Conv2D)	(None, 498, 498, 32)	320
max_pooling2d_6 (MaxPooling2D)	(None, 249, 249, 32)	0
conv2d_7 (Conv2D)	(None, 247, 247, 32)	9248
max_pooling2d_7 (MaxPooling2D)	(None, 123, 123, 32)	0
conv2d_8 (Conv2D)	(None, 121, 121, 32)	9248
max_pooling2d_8 (MaxPooling2D)	(None, 60, 60, 32)	0
conv2d_9 (Conv2D)	(None, 58, 58, 64)	18496
max_pooling2d_9 (MaxPooling2D)	(None, 29, 29, 64)	0
conv2d_10 (Conv2D)	(None, 27, 27, 64)	36928
max_pooling2d_10 (MaxPooling2D)	(None, 13, 13, 64)	0
conv2d_11 (Conv2D)	(None, 11, 11, 128)	73856
max_pooling2d_11 (MaxPooling2D)	(None, 5, 5, 128)	0

dropout_1 (Dropout)	(None, 5, 5, 128)	0
flatten_1 (Flatten)	(None, 3200)	0
dense_5 (Dense)	(None, 512)	1638912
dense_6 (Dense)	(None, 256)	131328
dense_7 (Dense)	(None, 128)	32896
dense_8 (Dense)	(None, 64)	8256
dense_9 (Dense)	(None, 2)	130

=====

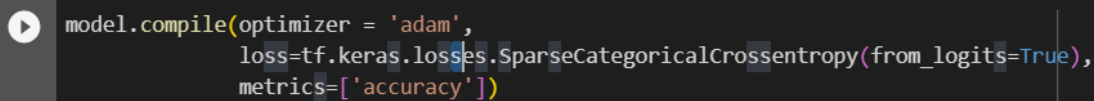
Total params: 1959618 (7.48 MB)
Trainable params: 1959618 (7.48 MB)
Non-trainable params: 0 (0.00 Byte)

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



```
model.compile(optimizer = 'adam',  
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
              metrics=['accuracy'])
```

Activity 4: Train the model

Now, let us train our model with our image dataset. The model is trained for 15 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 10 epochs and probably there is further scope to improve the model. **fit_generator** functions used to train a deep learning neural network

Arguments:

- **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. It specifies the total number of steps taken from the generator before it is stopped at every epoch and

its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

```
epochs= 15
history = model.fit_generator(
    train_data_gen,
    steps_per_epoch=int(np.ceil(total_train / float(Batch_Size))),
    epochs=epochs,
    validation_data = val_data_gen,
    validation_steps = int(np.ceil(total_val / float(Batch_Size)))
)
```

<ipython-input-39-96996318ca69>2: UserWarning: 'Model.fit_generator' is deprecated and will be removed in a future version. Please use 'Model.fit', which supports generators.

```
history = model.fit_generator(
Epoch 1/15
/usr/local/lib/python3.10/dist-packages/keras/src/backend.py:5729: UserWarning: ""sparse_categorical_crossentropy" received "from_logits=True", but the "output" argument was produce
output, from_logits = _get_logits(
29/29 [=====] - 23s 662ms/step - loss: 0.6887 - accuracy: 0.5497 - val_loss: 0.7571 - val_accuracy: 0.4566
Epoch 2/15
29/29 [=====] - 19s 633ms/step - loss: 0.6034 - accuracy: 0.6645 - val_loss: 0.6162 - val_accuracy: 0.6484
Epoch 3/15
29/29 [=====] - 21s 711ms/step - loss: 0.5117 - accuracy: 0.7660 - val_loss: 0.4056 - val_accuracy: 0.8311
Epoch 4/15
29/29 [=====] - 18s 639ms/step - loss: 0.5272 - accuracy: 0.7770 - val_loss: 0.5680 - val_accuracy: 0.8539
Epoch 5/15
29/29 [=====] - 19s 644ms/step - loss: 0.4646 - accuracy: 0.8079 - val_loss: 0.4435 - val_accuracy: 0.8082
Epoch 6/15
29/29 [=====] - 18s 631ms/step - loss: 0.4162 - accuracy: 0.8035 - val_loss: 0.4600 - val_accuracy: 0.8356
Epoch 7/15
29/29 [=====] - 18s 627ms/step - loss: 0.3544 - accuracy: 0.8477 - val_loss: 0.4725 - val_accuracy: 0.8584
Epoch 8/15
29/29 [=====] - 18s 622ms/step - loss: 0.3461 - accuracy: 0.8587 - val_loss: 0.3778 - val_accuracy: 0.8402
Epoch 9/15
29/29 [=====] - 18s 623ms/step - loss: 0.2978 - accuracy: 0.8499 - val_loss: 0.3409 - val_accuracy: 0.8219
Epoch 10/15
29/29 [=====] - 18s 632ms/step - loss: 0.2609 - accuracy: 0.8830 - val_loss: 0.2796 - val_accuracy: 0.9041
```

```
Epoch 9/15
29/29 [=====] - 18s 623ms/step - loss: 0.2978 - accuracy: 0.8499 - val_loss: 0.3409 - val_accuracy: 0.8219
Epoch 10/15
29/29 [=====] - 18s 632ms/step - loss: 0.2609 - accuracy: 0.8830 - val_loss: 0.2796 - val_accuracy: 0.9041
Epoch 11/15
29/29 [=====] - 20s 679ms/step - loss: 0.2121 - accuracy: 0.9117 - val_loss: 0.4654 - val_accuracy: 0.8174
Epoch 12/15
29/29 [=====] - 18s 617ms/step - loss: 0.3236 - accuracy: 0.8742 - val_loss: 0.2667 - val_accuracy: 0.8950
Epoch 13/15
29/29 [=====] - 18s 615ms/step - loss: 0.2478 - accuracy: 0.8852 - val_loss: 0.2661 - val_accuracy: 0.8950
Epoch 14/15
29/29 [=====] - 18s 628ms/step - loss: 0.1950 - accuracy: 0.9161 - val_loss: 0.2979 - val_accuracy: 0.9087
Epoch 15/15
29/29 [=====] - 18s 623ms/step - loss: 0.2495 - accuracy: 0.9007 - val_loss: 0.2371 - val_accuracy: 0.9178
```

From the above run time, we can easily observe that at 15th epoch the model is giving the better accuracy, which is initialized through call back parameter.

Activity 5 : Determining the testing accuracy of the Model

Model testing accuracy 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.

▾ Determining the accuracy of the model on Validation Data

```
[ ] test_accu = model.evaluate(val_data_gen)
    print('The testing accuracy is :',test_accu[1]*100, '%')
```

```
14/14 [=====] - 3s 219ms/step - loss: 0.2371 - accuracy: 0.9178
The testing accuracy is : 91.78082346916199 %
```

Activity 6 : Determining the testing accuracy of the Model

▾ Determining the accuracy of the model on Training Data

```
[ ] train_accu = model.evaluate(train_data_gen)
    print('The training accuracy is :',train_accu[1]*100, '%')
```

```
29/29 [=====] - 12s 400ms/step - loss: 0.1585 - accuracy: 0.9448
The training accuracy is : 94.48123574256897 %
```

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.

```
▶ model.save('covid.h5')
```

```
👤 /usr/local/lib/python3.10/dist-packages/keras/
   saving_api.save_model(
```

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

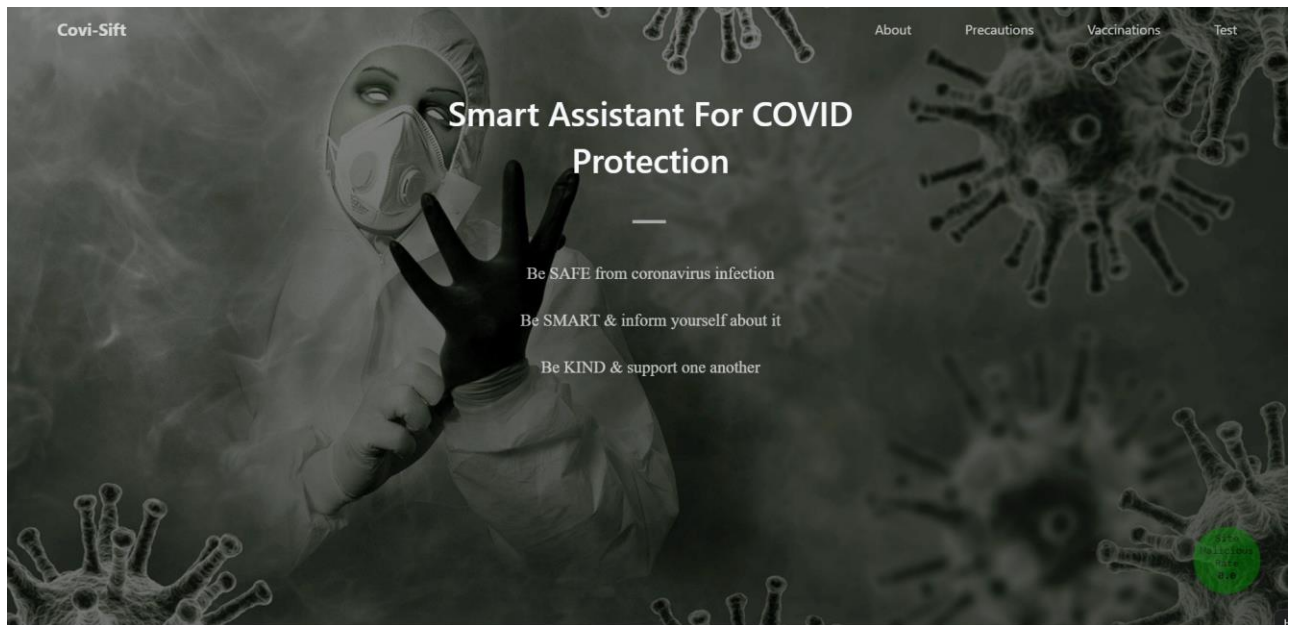
- Building HTML Pages
 - Building server side script

Activity1: Building Html Pages:

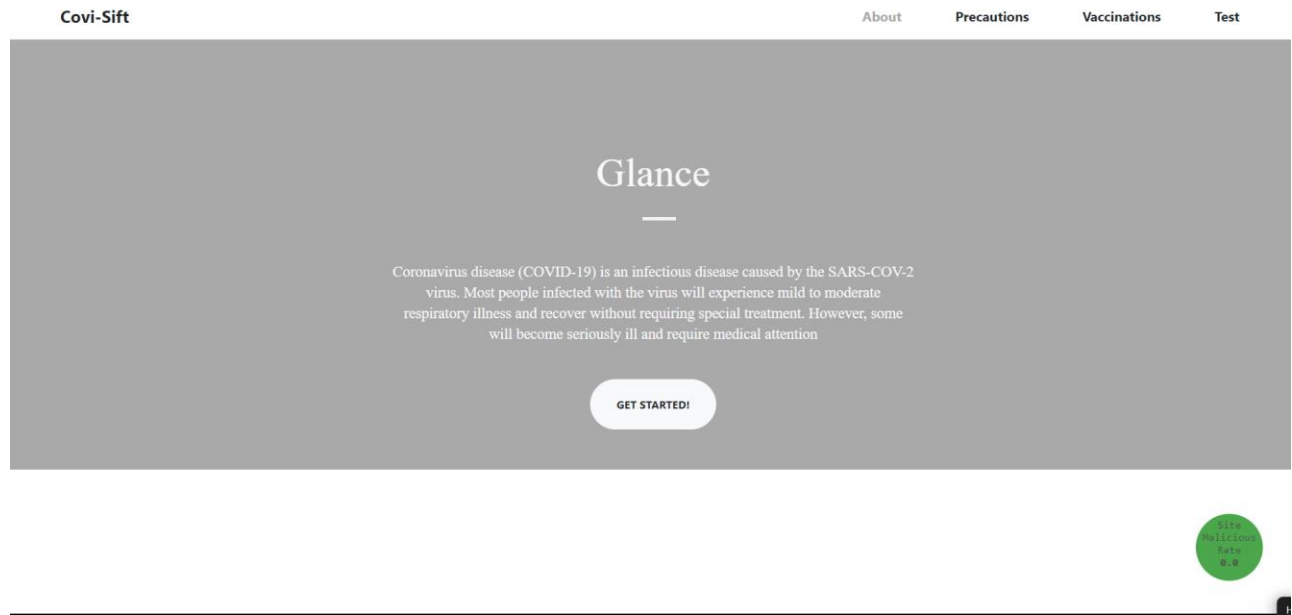
For this project create one HTML file namely

- Index.html

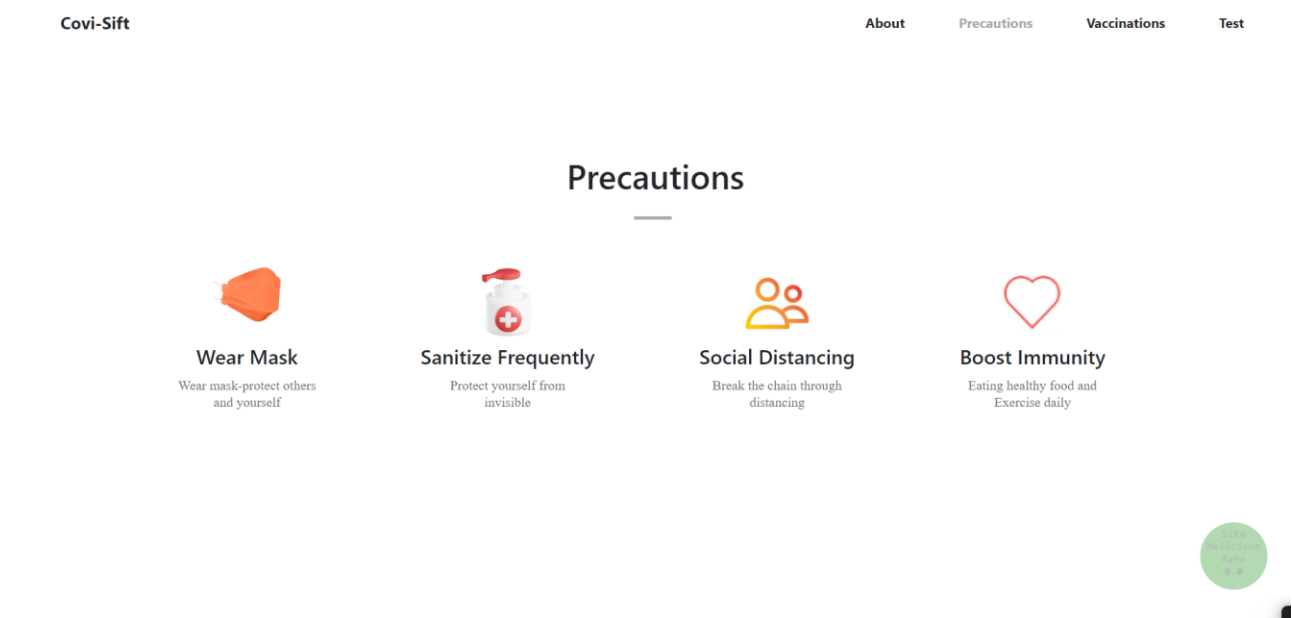
Let's see how our index.html page looks like:



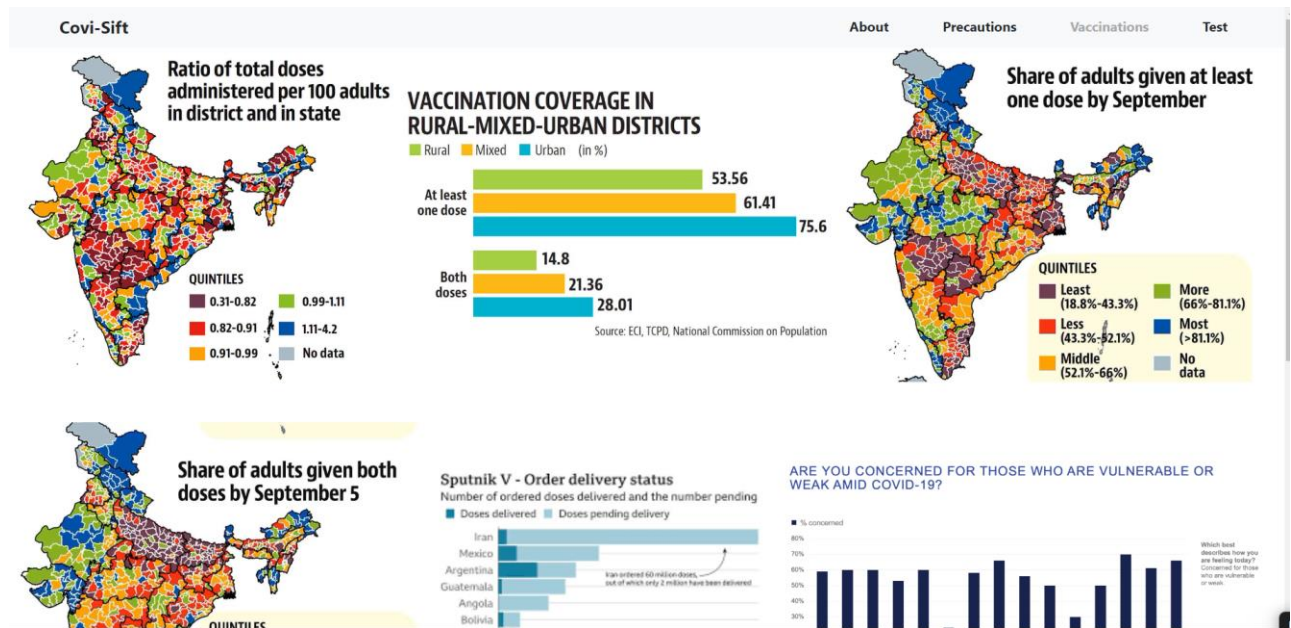
When you click on about button on the top , you will be redirecting to the following page



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



When you click on vaccinations, it will direct you the below page, where you can find the complete static information about the cases and vaccinations



When you click on the test button ,it will display the below page, where you can know more about our projects and you can the model by passing a image

Covi-Sift About Precautions Vaccinations Test

COVID-19 X-Ray Analysis

This page uses advanced machine learning techniques to analyze your X-ray image and determine whether you are at risk of having COVID-19.

Select your X-ray image:

Choose File No file chosen

Predict

Activity 2: Build Python code:

Import the libraries

```
import os
from flask import Flask, render_template, request
from keras.models import load_model
from keras.preprocessing import image
import numpy as np
```

Loading the saved model and initializing the flask app

```
app = Flask(__name__)

# Load the trained model
model = load_model('covid.h5')
```

Render HTML pages:

```
# Home route
@app.route('/')
def home():
    return render_template('Index.html')

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

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

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

@app.route('/Test')
def test():
    return render_template('Test.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.

```
# Predict route
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return render_template('Test.html', prediction_text='No file part')

    file = request.files['file']

    if file.filename == '':
        return render_template('Test.html', prediction_text='No selected file')

    if file and allowed_file(file.filename):
        # Load and preprocess the image
        img_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(img_path)
        img = image.load_img(img_path, target_size=(500, 500))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = np.expand_dims(img_array[:, :, :, 0], axis=-1)
        img_array /= 255.0

        # Make predictions
        predictions = model.predict(img_array)
        prediction = class_names[int(predictions[0, 0] > 0.5)]

        # Display the result on the web page
        return render_template('Test.html', prediction_text=f'The image is classified as: {prediction}')

    else:
        return render_template('Test.html', prediction_text='Invalid file format')
```

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.

Main Function:

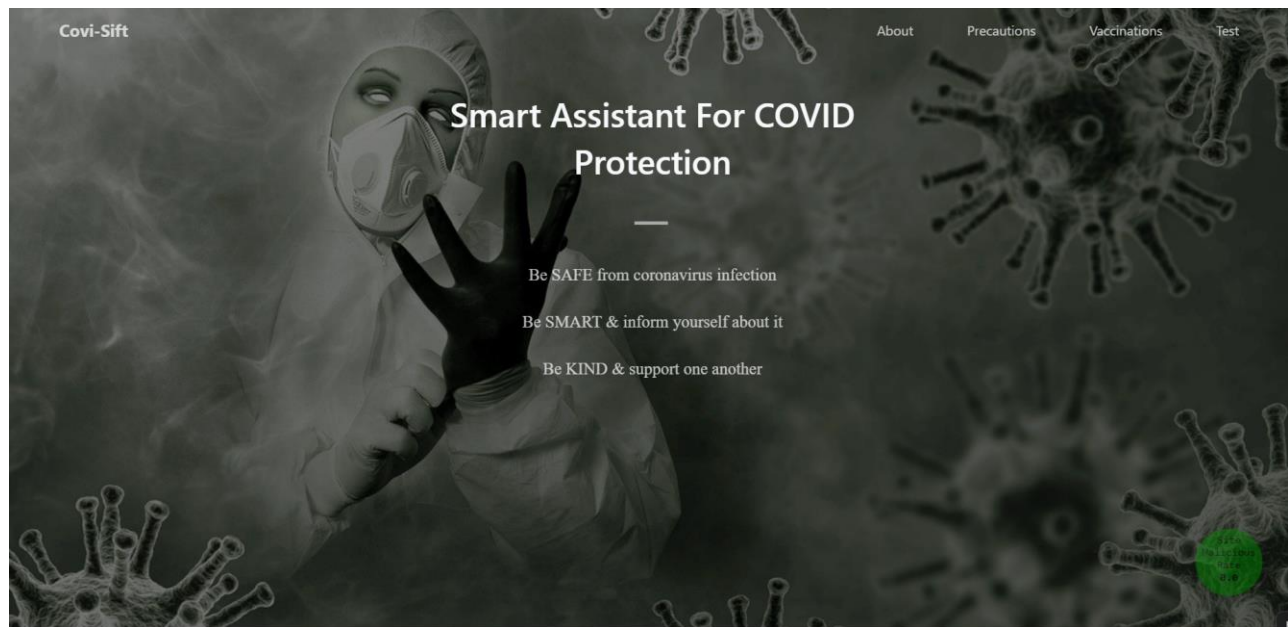
```
if __name__ == '__main__':
    #app.run(debug=True)
    app.run(host='0.0.0.0', port=5000, debug=True)
```

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.
-

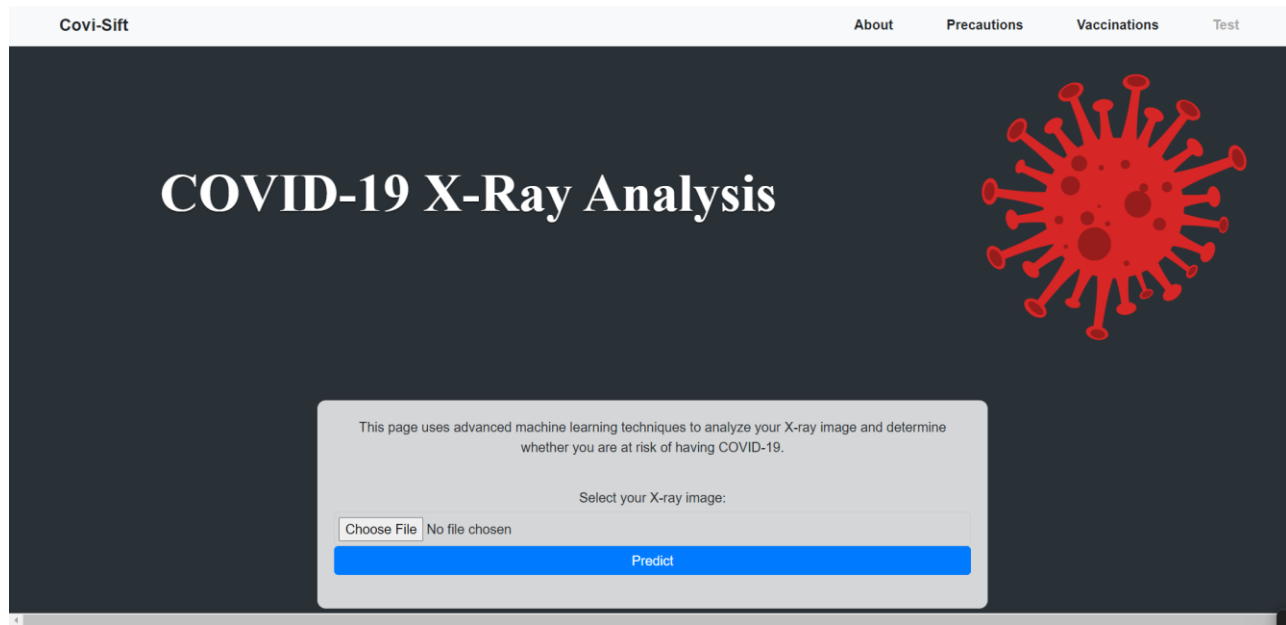
```
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.29.203:5000
Press CTRL+C to quit
* Restarting with stat
```

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



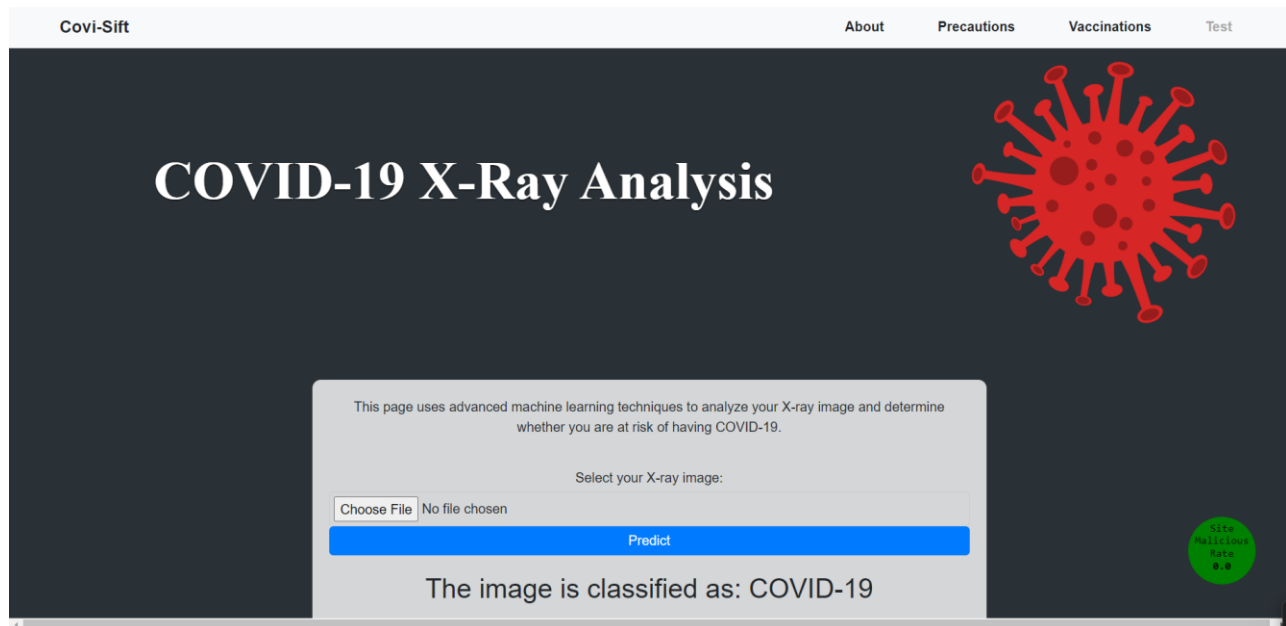
click on test button

Input 1:

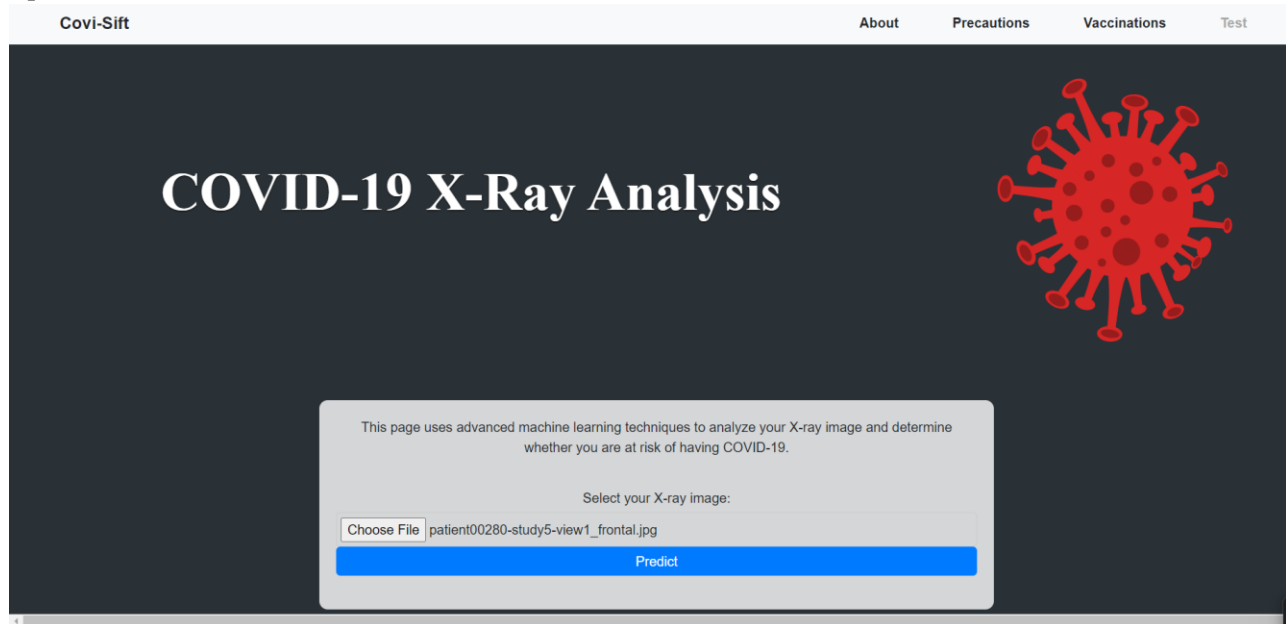


Once you upload the image and click on submit button, the output will be displayed in the below page

Output: 1



Input:2



Output:2

