

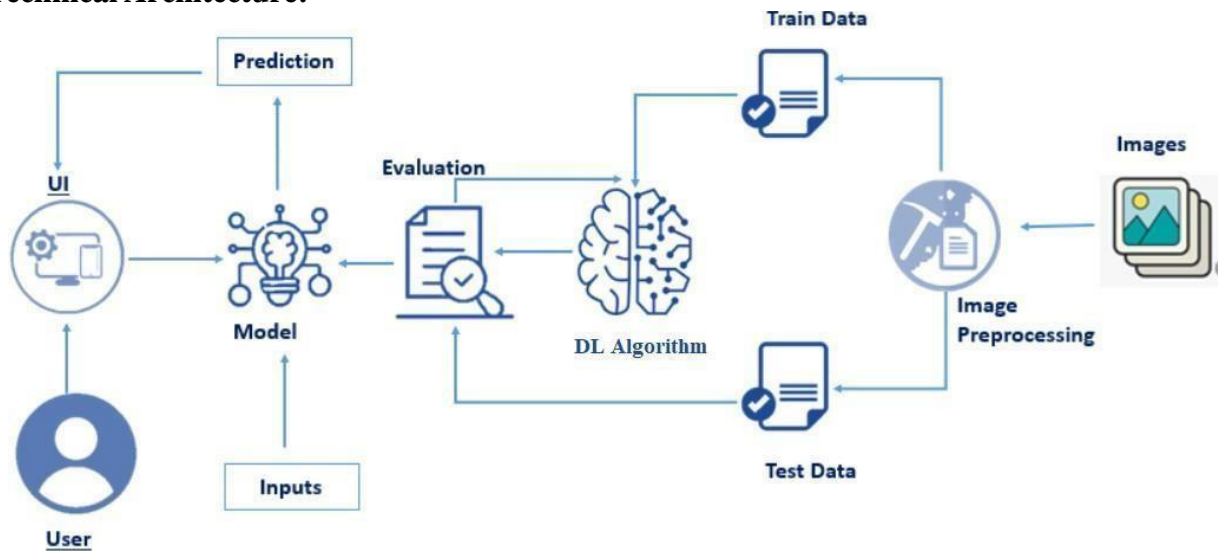
Detecting Covid-19 From Chest X-Rays Using Deep Learning Techniques

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 in 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 VGG16 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

- **Visual Studio Code / Google Colab:**

- Refer the link below to download anaconda navigator
- Link (Visual Studio Code)
https://www.youtube.com/watch?v=JPZsB_6yHVo&pp=ygUadmldzWFsIHN0dWRpbYBjb2RlIGluc3Rhbgw%3D
- Link (Google Colab) : <https://colab.research.google.com/>

- **Python packages:**

- Open terminal in vscode / write it with '!' before installing in Google Colab
- 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=MwZwr5Tvyxo&list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH>

Project Objectives:

By the end of this project you'll understand:

- Understanding Image Preprocessing Techniques
- Familiarity with Transfer Learning and CNN.
- Model Development and Training
- Evaluation Metrics and Model Validation.
- You will be able to Build web applications using the Flask framework.
- Deployment and Integration Skills
- User Interface Design for model Interface
- Comprehension of Healthcare AI Application

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 VGG16 Model analyzes the image, then the prediction is showcased on the Flask UI.

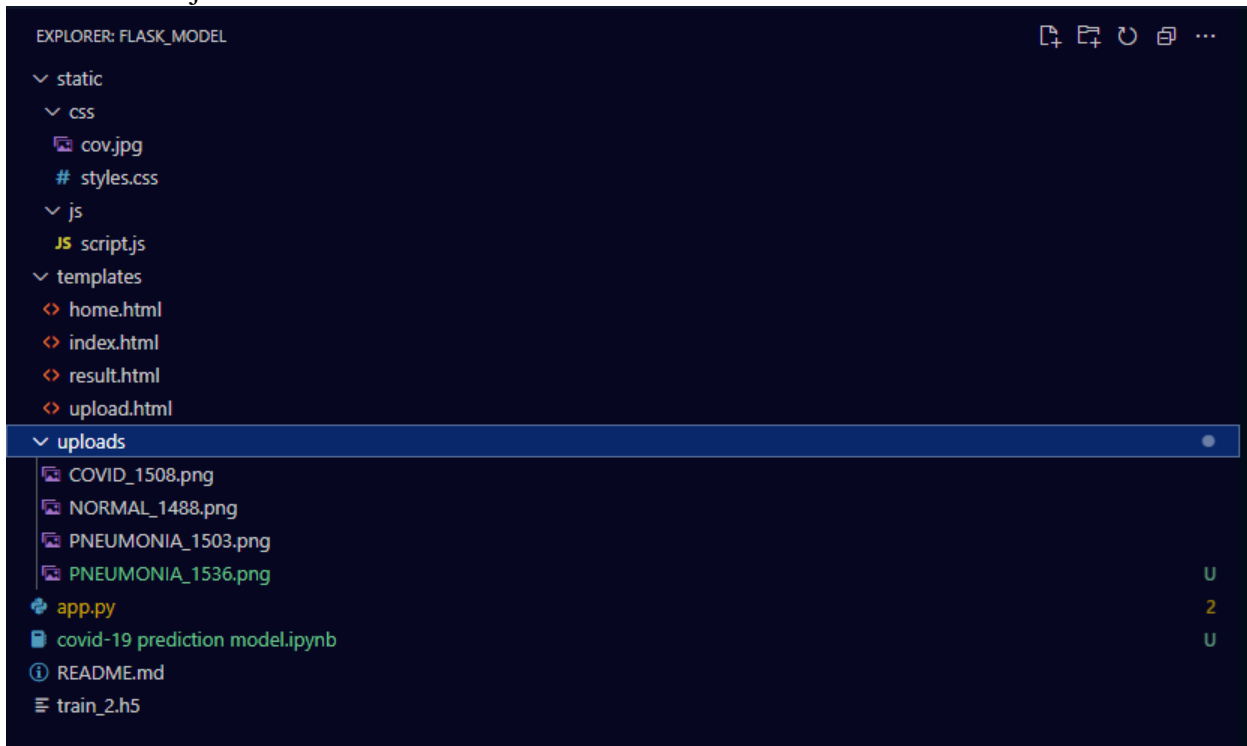
To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
 - Gather a Dataset containing chest X-ray images with labels for COVID-19, pneumonia, and normal cases
 - Organize the dataset into appropriate folders for train and test sets.
- Train and Test Data Preparation
 - Create separate paths for training and testing data from the collected dataset
 - Split the dataset into train and tests sets using 80-20 ratio.
- Data Pre-processing.
 - Import the libraries required to handle picture preprocessing and data augmentation, including ImageDataGenerator.
 - Preprocess the photos with ImageDataGenerator, including scaling, normalization, and augmentation procedures.

- Model development using VGG16
 - Set VGG16 as the feature extraction basis model. Adding Dense Layer
 - On top of the VGG16 layers, create a custom classification head (Dense layers).
 - Configure the Learning Process
 - Compile the model with appropriate optimizers, loss functions, and training metrics.
 - Save the Model
 - Test the model
- Training the Model
 - Utilize the preprocessed data generated by ImageDataGenerator for training
 - Train the VGG16 based model on the prepared train set.
 - Monitor and analyze model performance metrics during training iterations.
- Model Evaluation and Saving
 - Evaluate the trained VGG16 – based models performance using evaluation metrics on the test set.
 - Save the trained model's weights and architecture for the future development and inference.
- Flask and Model Integration
 - Set up a Flask application to serve as the backend for model integration.
 - Develop HTML files for user interaction and integrate with Flask backend.
 - Configure Flask route to handle image uploads and integrate the saved VGG16-based model for inference.
 - Implement Python code in Flask routes to preprocess uploaded images and make predictions using the trained model.

Project Structure:

Create a Project folder which contains files as shown below



- Flask folder consists of static, templates and app.py
- Covid-19 prediction model is the VGG16-based model
- Train_2.h5 is the trained and saved model

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/datasets/sachinkumar413/covid-pneumonia-normal-chest-xray-images>

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

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

Once mounted the drive create a folder with project name and move the dataset to that folder and mount to that folder.

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 (Xception) is selected. The image input size of xception model is 299, 299.

```
[ ] train_path = "/content/drive/MyDrive/Covid_data/Model_data/train"
    test_path = "/content/drive/MyDrive/Covid_data/Model_data/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.

Activity 1: Importing the libraries

Import the necessary libraries as shown in the image

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Sequential
import tensorflow as tf
import numpy as np
import cv2
```

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 shear via shear_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)
```

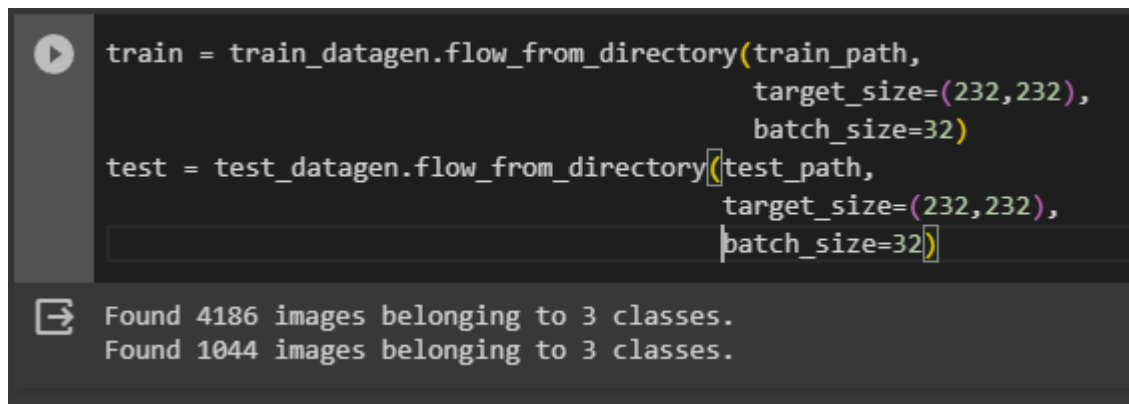
Activity 3: Apply ImageDataGenerator functionality to Train set and Test set

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).



```
train = train_datagen.flow_from_directory(train_path,
                                         target_size=(232,232),
                                         batch_size=32)
test = test_datagen.flow_from_directory(test_path,
                                       target_size=(232,232),
                                       batch_size=32)
```

Found 4186 images belonging to 3 classes.
Found 1044 images belonging to 3 classes.

Total the dataset is having 19159 train images and 2006 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 Xception, one of the convolution neural net (CNN) architecture which is considered as a very good model for Image classification.

Deep understanding on the Xception model – Link is referred to in the prior knowledge section. Kindly refer to it before starting the model building part.

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 (299,299,3).

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.

```
[ ] conv_base = VGG16(weights='imagenet',include_top =False,input_shape=(232,232,3))  
  
▶ conv_base.summary()
```

Activity 2: Adding Dense Layers

```
▶ model = Sequential()  
  model.add(conv_base)  
  model.add(Flatten())  
  model.add(Dense(256,activation='relu'))  
  model.add(Dense(3,activation='softmax'))
```

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 from `vgg_model` and output as dense layer.

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.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6422784
dense_1 (Dense)	(None, 3)	771

=====
Total params: 21138243 (80.64 MB)
Trainable params: 6423555 (24.50 MB)
Non-trainable params: 14714688 (56.13 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

```
#compiling the mode.  
model.compile(loss="categorical_crossentropy",  
              optimizer="adam",  
              metrics="accuracy")
```

Activity 4: Train the model

Now, let us train our model with our image dataset. The model is trained for 25 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.

```
class StopTrainingCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('val_accuracy') >= 0.990:
            print("\nValidation accuracy reached 99.0% or higher. Stopping training.")
            self.model.stop_training = True

[ ] #fit the model
model.fit(train, epochs=10,
          validation_data = test,
          validation_steps= len(test)//32,
          callbacks=[StopTrainingCallback()])
model.save('/content/drive/MyDrive/Covid data/Models/train_2.h5')
```

```
Epoch 1/10
131/131 [=====] - 1800s 14s/step - loss: 0.2812 - accuracy: 0.8982 - val_loss: 0.3894 - val_accuracy: 0.8750
Epoch 2/10
131/131 [=====] - 1798s 14s/step - loss: 0.1477 - accuracy: 0.9522 - val_loss: 0.4043 - val_accuracy: 0.9062
Epoch 3/10
131/131 [=====] - ETA: 0s - loss: 0.1179 - accuracy: 0.9651
Validation accuracy reached 80.0% or higher. Stopping training.
131/131 [=====] - 1799s 14s/step - loss: 0.1179 - accuracy: 0.9651 - val_loss: 0.0737 - val_accuracy: 1.0000
```

From the above run time, we can easily observe that at 9th epoch the model is giving the 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.

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

[ ] from tensorflow.keras.models import load_model
model = load_model('/content/drive/MyDrive/Covid data/Models/train_2.h5')

[ ] print(train.class_indices)

{'COVID': 0, 'NORMAL': 1, 'PNEUMONIA': 2}

[ ] img = image.load_img('/content/drive/MyDrive/Covid data/Model_data/test/PNEUMONIA/PNEUMONIA_1449.png',target_size=(232,232))
x = image.img_to_array(img)
x = np.expand_dims(x,axis=0)
pred = model.predict(x)
pred_class = np.argmax(pred, axis=1)
index = ['COVID','NORMAL','PNEUMONIA']
str(index[pred_class[0]])

1/1 [=====] - 0s 396ms/step
'PNEUMONIA'
```

- In the above code, we have tested the model with a image of x-rays, which comes under the COVID label

- Model which us trained is working better with unknown data also, 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.

```
callbacks=[StopTrainingOnNaNValLoss()],  
model.save('/content/drive/MyDrive/Covid data/Models/train_2.h5')
```

Accuracy Metrics

Testing

```
[ ] test_loss, test_accuracy = model.evaluate(test)

33/33 [=====] - 444s 13s/step - loss: 0.1425 - accuracy: 0.9521

▶ print(f'Test accuracy: {test_accuracy* 100:.2f}%')

📄 Test accuracy: 95.21%
```

95.21%

Training accuracy

```
[ ] _, train_accuracy = model.evaluate(train)

131/131 [=====] - 1779s 14s/step - loss: 0.0756 - accuracy: 0.9768
```

97.68%

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/CSS file namely

- home.html
- upload.html
- result.html
- styles.css

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



COVID-19 X-Ray Analyzer

History

Press **F11** to exit full screen

The outbreak of COVID-19, arising in December 2019 in Wuhan, China, initially manifested as a cluster of pneumonia cases with an unknown cause. The novel coronavirus, later identified as SARS-CoV-2, swiftly evolved into a global pandemic, prompting the World Health Organization (WHO) to declare it as such on March 11, 2020. Its rapid transmission through respiratory droplets and close contact raised significant concerns regarding public health and safety. The virus posed substantial hazards due to its high infectivity rate and the potential for causing severe respiratory illness, impacting populations worldwide. The urgency to address this health crisis and mitigate its devastating effects prompted a search for innovative solutions to manage and combat the spread of the virus.

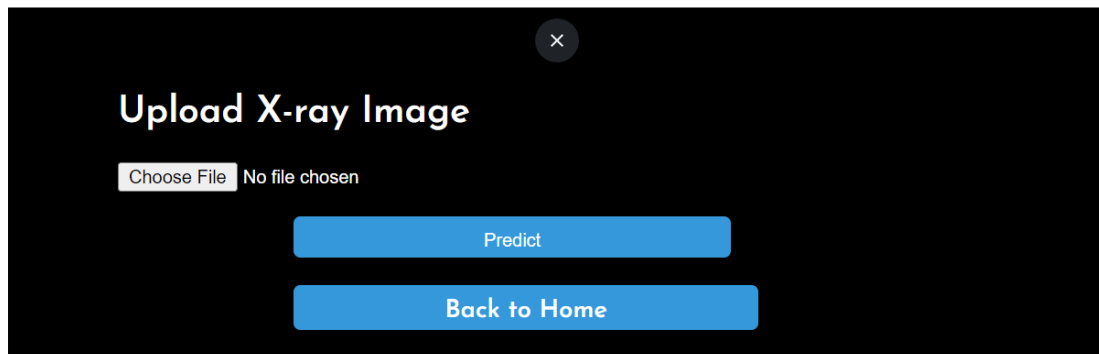
Amid the escalating pandemic, the pressing need for effective and prompt identification of COVID-19 cases became evident. Traditional diagnostic methods, particularly polymerase chain reaction (PCR) tests, while accurate, presented challenges due to their time-consuming nature, taking several hours or longer to yield results. Delays in obtaining test results hindered timely patient triage and treatment, exacerbating the strain on healthcare systems already grappling with a surge in cases. In response to these challenges, the integration of medical imaging, specifically chest X-rays, alongside artificial intelligence emerged as a pivotal approach to expedite the assessment and management of COVID-19-infected individuals.

In this context, the utilization of deep-learning techniques within artificial intelligence gained prominence for their potential in rapidly analyzing chest X-ray images to detect COVID-19-related lung abnormalities. These advanced AI algorithms offered high-performance classification capabilities, swiftly identifying pathological findings indicative of the virus in chest X-rays. Leveraging AI-based analysis of medical images allowed for a more efficient and accurate triage process, enabling healthcare professionals to swiftly differentiate between patients with normal chest X-ray results and those displaying COVID-19-related lung pathologies. This innovative approach significantly aided in alleviating the burden on emergency departments and urgent care centers, facilitating quicker decision-making regarding hospital admission for patients requiring close monitoring and medical intervention. By integrating deep-learning models into the diagnostic pathway, healthcare providers could expedite the identification and management of COVID-19 cases, contributing to more effective containment strategies during the ongoing global health crisis.

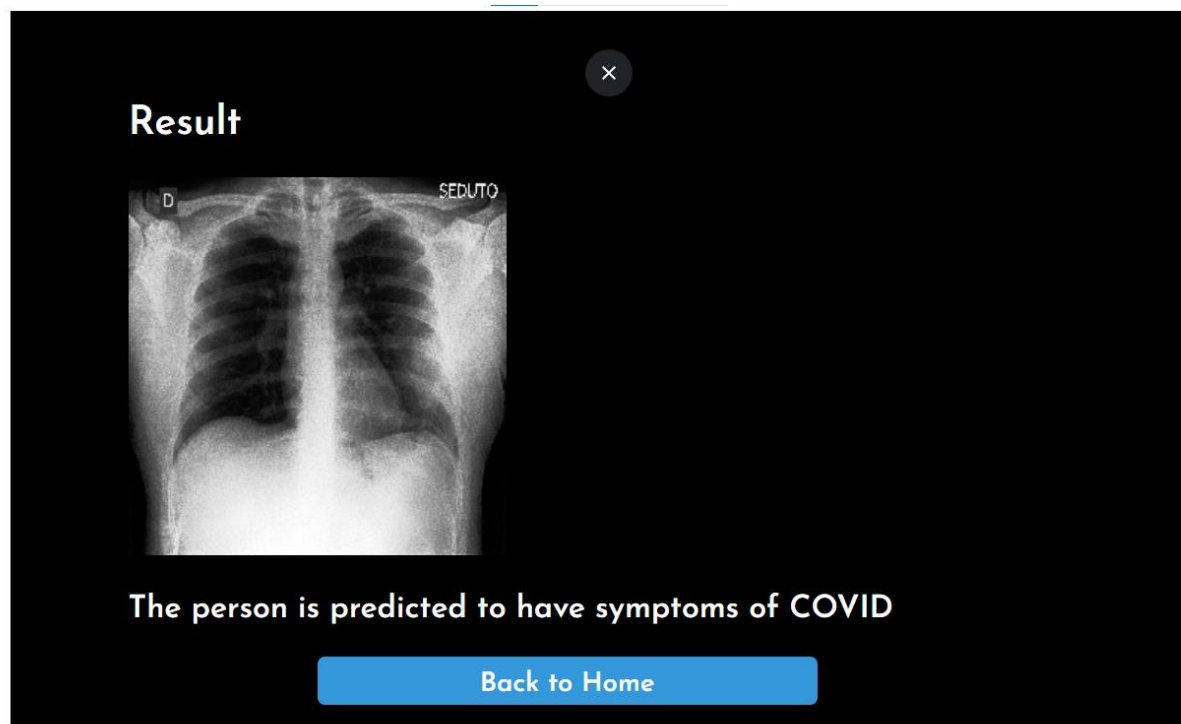
Welcome to the COVID-19 X-ray Analyzer. Upload an X-ray image to get started.

[Go to Upload Page](#)

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

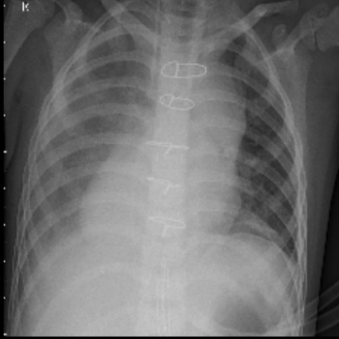


When you click on the predict button after choosing the x-ray, it will redirect you according to the prediction obtained.



Press **F11** to exit full screen

Result



The person is predicted to have symptoms of **PNEUMONIA**

[Back to Home](#)

×

Press **F11** to exit full screen

Result



The Person has no symptoms of any disease and is predicted as **NORMAL**

[Back to Home](#)

Activity 2: Build Python code:

Importing the libraries

```
1 from flask import Flask , render_template , request, send_from_directory, url_for
2 import numpy as np
3 from tensorflow.keras.models import load_model
4 from tensorflow.keras.preprocessing import image
5 import os
6
```

Loading the saved model and initializing the flask app

```
app = Flask(__name__)

model = load_model('train_2.h5' , compile = False)
```

Render HTML pages:

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

@app.route('/upload/<path:filename>')
def get_image(filename):
    return send_from_directory('uploads', filename)

@app.route('/result' , methods=['POST'])
def result():
    image_path = request.args.get('image_path')
    return render_template('result.html', image_path=image_path)
```

After we submitted the file into the programme, we checked to see if it was properly uploaded. Here, we will use the specified constructor to route to the HTML page that we constructed earlier.

In the preceding example the '/' URL is associated with the home.html function. As a result , when the web server's home page is opened in a browser, the HTML page is rendered. When you enter the values can be obtained using the POST method

```

17 @app.route('/upload', methods=['GET', 'POST'])
18 def upload():
19     if request.method == 'POST':
20         if 'image' in request.files:
21             f = request.files['image']
22             if f.filename != '':
23                 # Specify the directory to save uploaded images
24                 upload_folder = 'uploads'
25
26                 # Ensure the upload directory exists; if not, create it
27                 if not os.path.exists(upload_folder):
28                     os.makedirs(upload_folder)
29
30                 # Save the uploaded image to the specified directory
31                 image_path = os.path.join(upload_folder, f.filename)
32                 f.save(image_path)
33
34                 img = image.load_img(image_path, target_size= (232,232))
35                 img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # converting the uploaded image to gray-scale
36                 x = image.img_to_array(img)
37                 print(x)
38                 x=np.expand_dims(x, axis=0)
39                 print(x)
40                 y = model.predict(x)
41                 preds = np.argmax(y, axis=1)
42
43                 print('prediction', preds)
44
45                 index = ['COVID', 'NORMAL', 'PNEUMONIA']
46                 if preds[0]==0:
47                     statement = "The person is predicted to have symptoms of "
48                 elif preds[0]==1:
49                     statement = "The Person has no symptoms of any disease and is predicted as "
50                 else:
51                     statement="The person is predicted to have symptoms of "
52
53                 text = statement + str(index[preds[0]])
54                 print(str(index[preds[0]]))
55
56                 # Pass the image path to the 'result.html' template for display
57                 return render_template('result.html', image_path=f.filename, text=text)
58
59     return render_template('upload.html')

```

Here, we're directing our app to the res function. Using a Post request, this function obtains all of the values from the HTML page. This information is saved in an array. The model.predict() function receives this array. The prediction is returned by this function. And this prediction value will be rendered to the text mentioned earlier in the index.html page..

Main Function:

```

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

```

Activity 3: Run the application

- Open VScode
- 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.

```
PS C:\Users\jayvinit\Desktop\Group_Project_Smart_Internz\Flask_model> & C:/Users/jayvinit/AppData/Local/Programs/Python/Python311/python.exe c:/Users/jayvinit/Desktop/Group_Project_Smart_Internz/Flask_model/app.py
2023-11-21 16:43:23.713538: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow 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. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
2023-11-21 16:43:33.561664: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
* Debugger is active!
* Debugger PIN: 432-636-861
```