Covid-19 Detection from Lung X-rays

Project Description:

The global outbreak of COVID-19 has emphasized the need for swift and accurate diagnostic tools to curb the spread of the virus. This project addresses this challenge by leveraging medical images, specifically chest X-rays, and employing advanced artificial intelligence (AI) techniques for rapid assessment and detection of COVID-19.

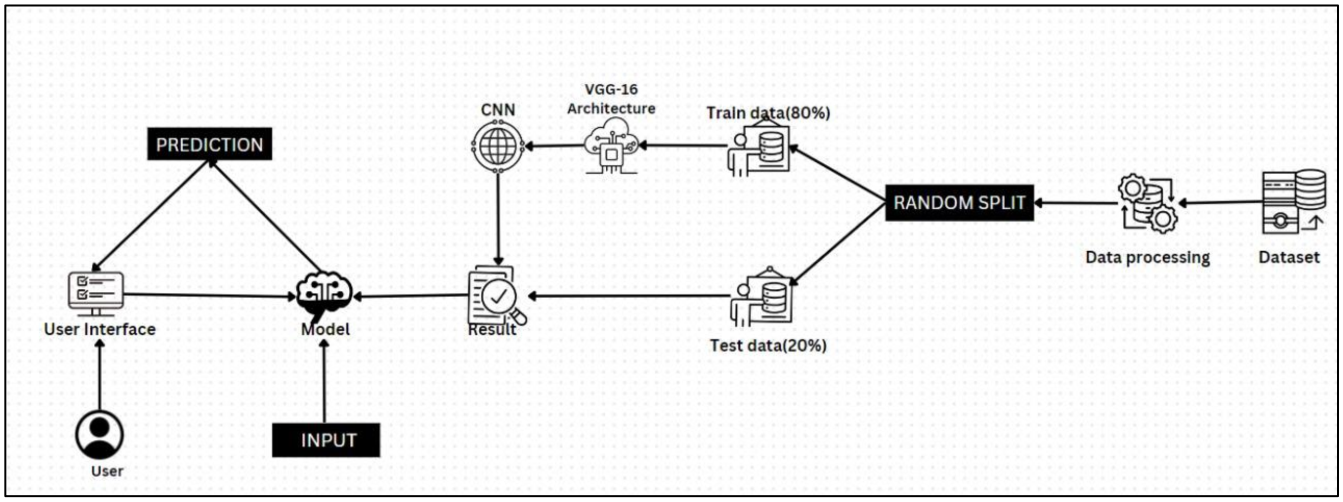
As the pandemic continues to evolve, the demand for efficient diagnostic solutions becomes increasingly critical. Traditional diagnostic methods, such as PCR tests, often come with a time delay in providing results. In contrast, the analysis of chest X-ray images offers a rapid and valuable complement to clinical assessments. A normal chest X-ray may guide patients toward home care while awaiting confirmatory test results, while pathological findings can prompt immediate hospital admission for close monitoring.

This project focuses on the application of deep learning (DL) methods, a subset of AI, to develop a high-performance classifier for the detection of COVID-19 using chest X-rays. DL algorithms, particularly transfer learning techniques like Inception V3, ResNet50, and Xception V3, have demonstrated significant success in medical image analysis and classification. By training our model on diverse datasets encompassing both COVID-19 positive and negative cases, we aim to create a robust and accurate tool for early diagnosis.

The key objectives of the project are **Data Collection**, **Data Preprocessing, Transfer Learning, Model Training, Validation and Testing, User Interface Implementation, Ethical Considerations, Impact Assessment.**

By combining cutting-edge AI techniques with medical imaging, this project aims to contribute significantly to the timely and accurate detection of COVID-19, ultimately supporting healthcare professionals in their efforts to manage and mitigate the impact of the pandemic.

Solution Architecture:



Prerequisites:

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

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.
- Type "pip install pillow" 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:

- Image Preprocessing:

Gain proficiency in preprocessing medical images, enhancing their quality, and standardizing formats.

- Transfer Learning Implementation:

Apply transfer learning algorithms to the dataset, utilizing pre-trained models for effective feature extraction.

- Understanding Deep Neural Networks:

Develop insights into the functioning of deep neural networks and their application in disease detection.

- Model Accuracy Assessment:

Learn to evaluate the accuracy of the model, considering key metrics for performance assessment.

- Web Application Development with Flask:

Acquire the skills to build web applications using the Flask framework, facilitating user-friendly interfaces for model deployment.

Project Flow:

- User Interaction:

Users engage with the UI to select an image for analysis.

- Model Integration with Flask:

The chosen image is processed by the integrated Xception Model within the Flask application.

- Analysis and Prediction Display:

The Xception Model analyzes the image, and the resulting prediction is presented on the Flask UI.

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

1. **Data Collection:**
    - Establish paths for training and testing datasets.

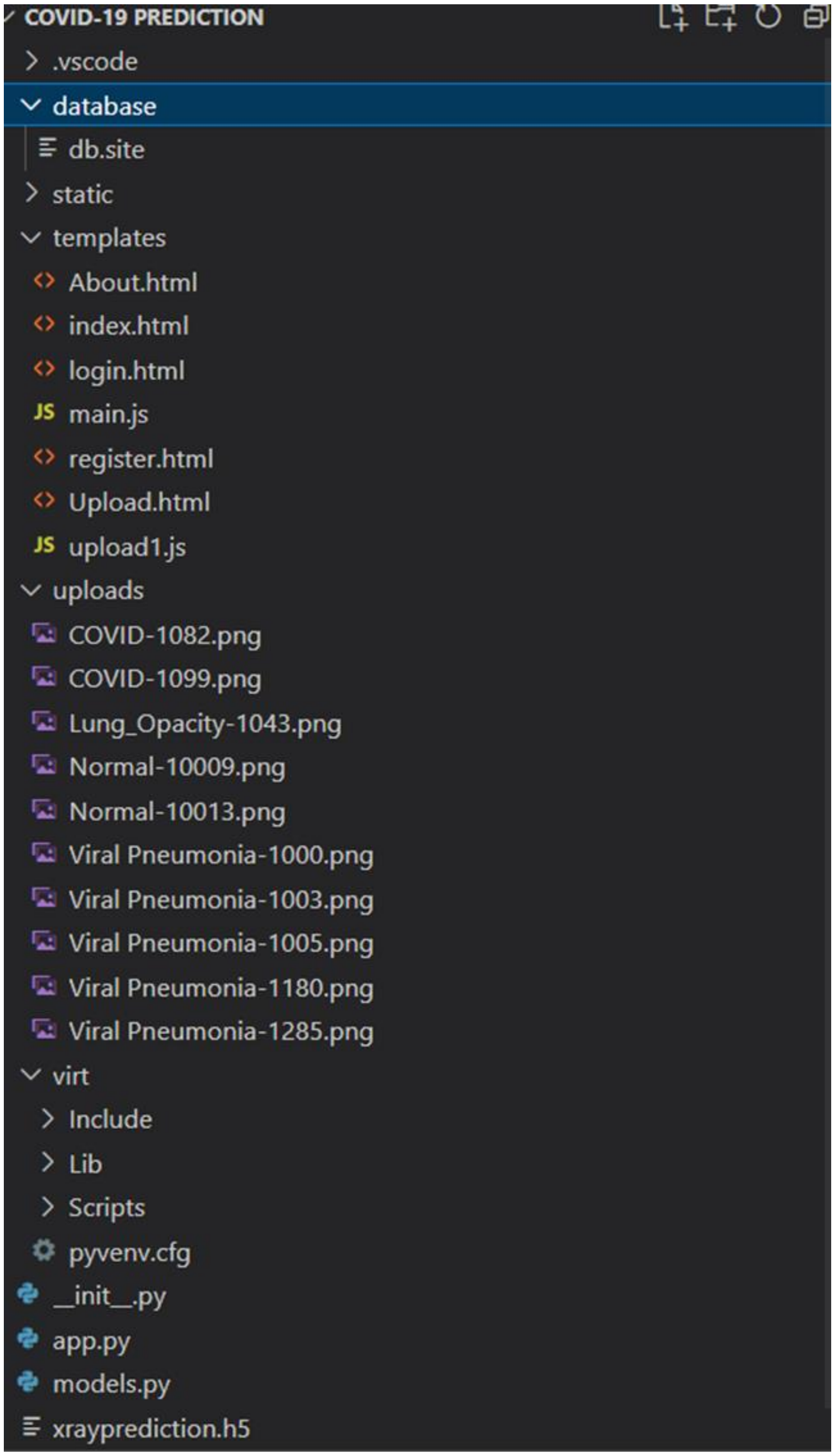
2. **Data Pre-processing:**
    - Import necessary libraries.
    - Set up the ImageDataGenerator class.
    - Apply ImageDataGenerator functionality to preprocess the training and testing sets.

3. **Model Building:**
    - Utilize a pre-trained CNN model as a feature extractor.
    - Integrate a Dense Layer for classification.
    - Configure the learning process.
    - Train the model on the prepared dataset.
    - Save the trained model.
    - Test the model's performance.

4. **Application Building:**
    - Create an HTML file for the user interface.
    - Develop the Python code to implement the application.

Project Structure:

Create a Project folder which contains files as shown below



```
COVID-19 PREDICTION
> .vscode
v database
    db.site
> static
v templates
    About.html
    index.html
    login.html
    main.js
    register.html
    Upload.html
    upload1.js
v uploads
    COVID-1082.png
    COVID-1099.png
    Lung_Opacity-1043.png
    Normal-10009.png
    Normal-10013.png
    Viral Pneumonia-1000.png
    Viral Pneumonia-1003.png
    Viral Pneumonia-1005.png
    Viral Pneumonia-1180.png
    Viral Pneumonia-1285.png
v virt
  > Include
  > Lib
  > Scripts
    pyvenv.cfg
  __init__.py
  app.py
  models.py
  xrayprediction.h5
```

- Virtual environment is used to run the files
- Database is to store information of the patients.
- Uploads folder contains the uploaded images
- Models' folder contains the code of database connection
- App contains the integration of model through flask and routing

Step 1: Data Collection

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

Download the dataset

Aggregate a collection of chest X-ray images capturing various Covid-19 scenarios, such as Covid-19 positive, Normal, Viral Pneumonia, and Lung Opacity. Systematically arrange these images into specific subdirectories, aligning with their designated categories as outlined in the project structure. Establish folders for each distinct type of Covid-19, facilitating the recognition process within the project.

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

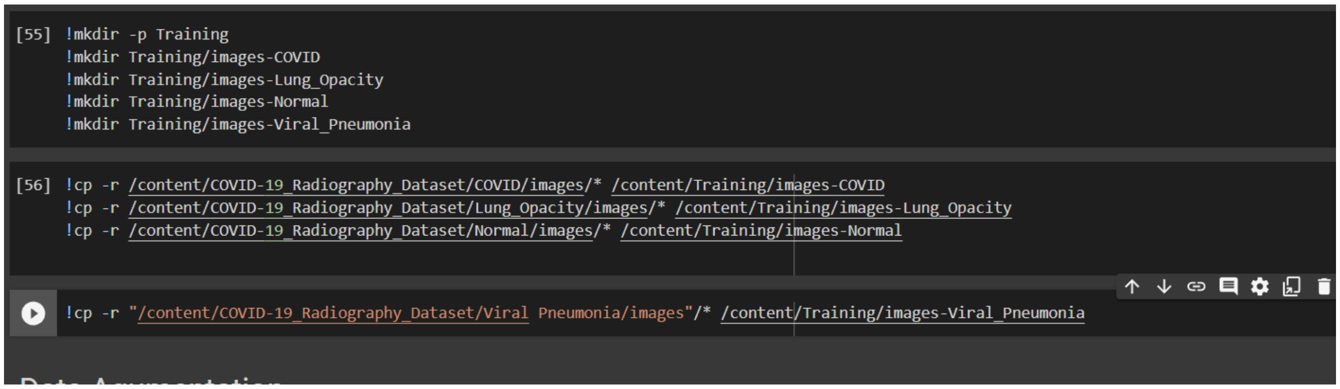
Dataset:- **https://www.kaggle.com/code/rollanmaratov/covid19-detection-using-tensorflow-from-chest-xray/data**

Note: For better accuracy train on more images

We will be constructing our training model on Google Colab.
To facilitate this process, upload the dataset to Google Drive and establish a connection between Google Colab and Drive using the provided code snippet.
Upon successful mounting of the drive, generate a dedicated folder named after the project and transfer the dataset into this folder.
Subsequently, link the Colab session to this designated project folder for seamless access and utilization.

Extracting dataset for training

To construct a Deep Learning (DL) model, it's essential to segregate training in a folders. We have extracted all the images form the dataset into folders created with their respective names. This data is later split for training and testing.



```
[55] !mkdir -p Training
     !mkdir Training/images-COVID
     !mkdir Training/images-Lung_Opacity
     !mkdir Training/images-Normal
     !mkdir Training/images-Viral_Pneumonia

[56] !cp -r /content/COVID-19_Radiography_Dataset/COVID/images/* /content/Training/images-COVID
     !cp -r /content/COVID-19_Radiography_Dataset/Lung_Opacity/images/* /content/Training/images-Lung_Opacity
     !cp -r /content/COVID-19_Radiography_Dataset/Normal/images/* /content/Training/images-Normal

     !cp -r "/content/COVID-19_Radiography_Dataset/Viral Pneumonia/images"/* /content/Training/images-Viral_Pneumonia
```

Step 2: Image Preprocessing

Image preprocessing refers to a set of techniques applied to raw image data before feeding it into a machine learning or computer vision model. The goal is to enhance the quality, remove noise, and extract relevant features, thereby improving the model's performance in tasks such as image classification, object detection, or segmentation. Common preprocessing steps include resizing, normalization, data augmentation, and denoising, among others. The aim is to optimize the input data to facilitate more effective learning and generalization by the model.
.

Configure ImageDataGenerator class

The ImageDataGenerator class is initialized, configuring various data augmentation techniques for image data. The key augmentation methods include:

Image Shifts:
- Implemented through the width_shift_range and height_shift_range arguments.

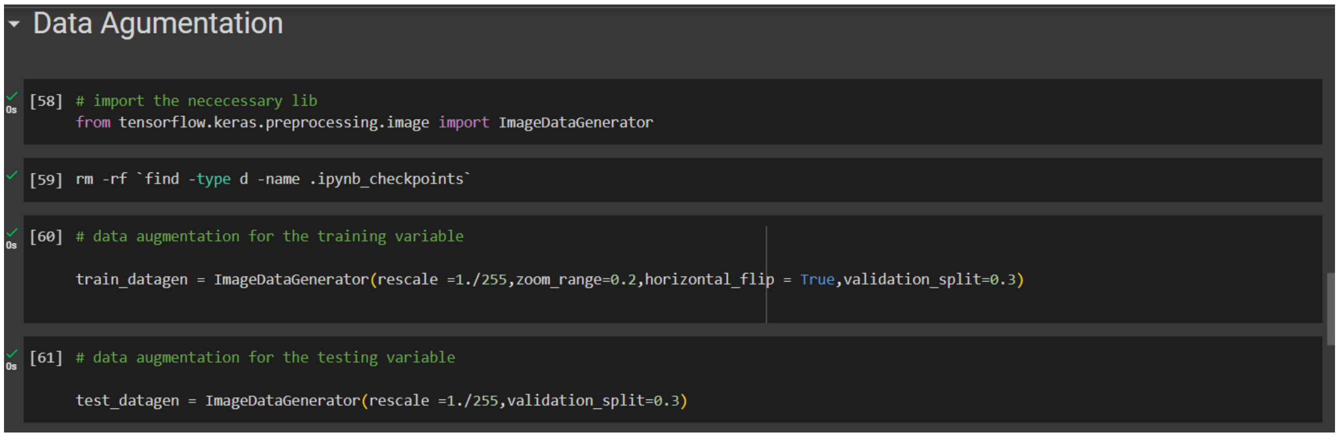
Image Flips:
- Achieved using the horizontal_flip and vertical_flip arguments.

Image Rotations:
- Applied through the rotation_range argument.

Image Brightness Adjustment:
- Controlled by the brightness_range argument.

Image Zoom:
- Controlled through the zoom_range argument.

These techniques collectively contribute to the augmentation process, enhancing the diversity and robustness of the dataset for improved model training.An instance of the ImageDataGenerator class can be constructed for train and test.



Data Agumentation

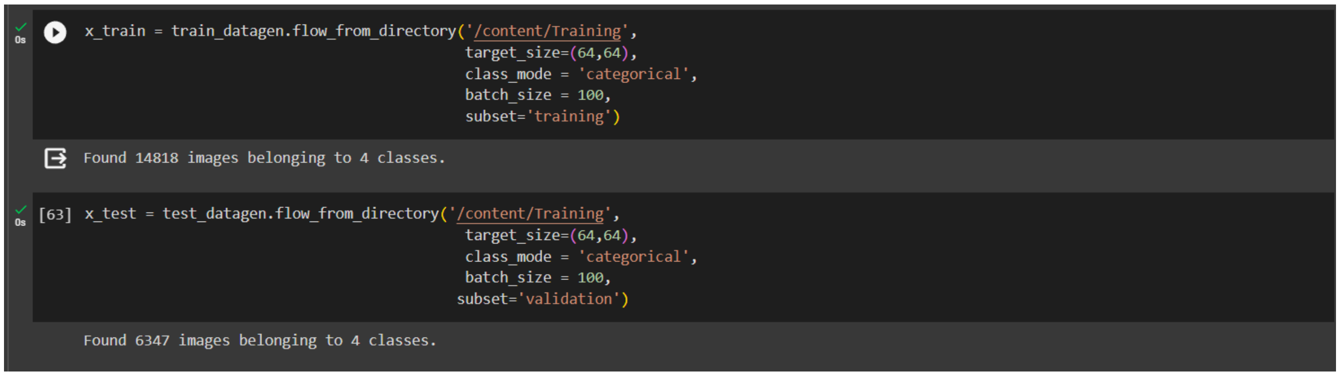
```
[58] # import the nececessary lib
     from tensorflow.keras.preprocessing.image import ImageDataGenerator

[59] rm -rf `find -type d -name .ipynb_checkpoints`

[60] # data augmentation for the training variable

     train_datagen = ImageDataGenerator(rescale =1./255,zoom_range=0.2,horizontal_flip = True,validation_split=0.3)

[61] # data augmentation for the testing variable

     test_datagen = ImageDataGenerator(rescale =1./255,validation_split=0.3)
```

Apply ImageDataGenerator functionality to Train set and Test set

Let's implement the ImageDataGenerator functionality for both the Training and Test sets using the provided code.

Specifically, for the Training set, the flow_from_directory function is employed, returning image batches from the specified directory. The relevant arguments for this function include:

- **directory:** The location of the data. If labels are "inferred," it should comprise subdirectories, each representing a class with its respective images. Otherwise, the directory structure is disregarded.
- **batch_size**: The size of the data batches, set to 32.
- **target_size:** The dimensions to resize images to after reading them from disk.
- **class_mode:**
    - 'int': Implies that labels are encoded as integers (e.g., for sparse_categorical_crossentropy loss).
    - 'categorical': Implies that labels are encoded as categorical vectors (e.g., for categorical_crossentropy loss).
    - 'binary': Signifies that there are only two labels, encoded as float32 scalars with values 0 or 1 (e.g., for binary_crossentropy).
    - None: Indicates no labels are present.



```
x_train = train_datagen.flow_from_directory('/content/Training',
                                            target_size=(64,64),
                                            class_mode = 'categorical',
                                            batch_size = 100,
                                            subset='training')

Found 14818 images belonging to 4 classes.

[63] x_test = test_datagen.flow_from_directory('/content/Training',
                                            target_size=(64,64),
                                            class_mode = 'categorical',
                                            batch_size = 100,
                                           subset='validation')

Found 6347 images belonging to 4 classes.
```

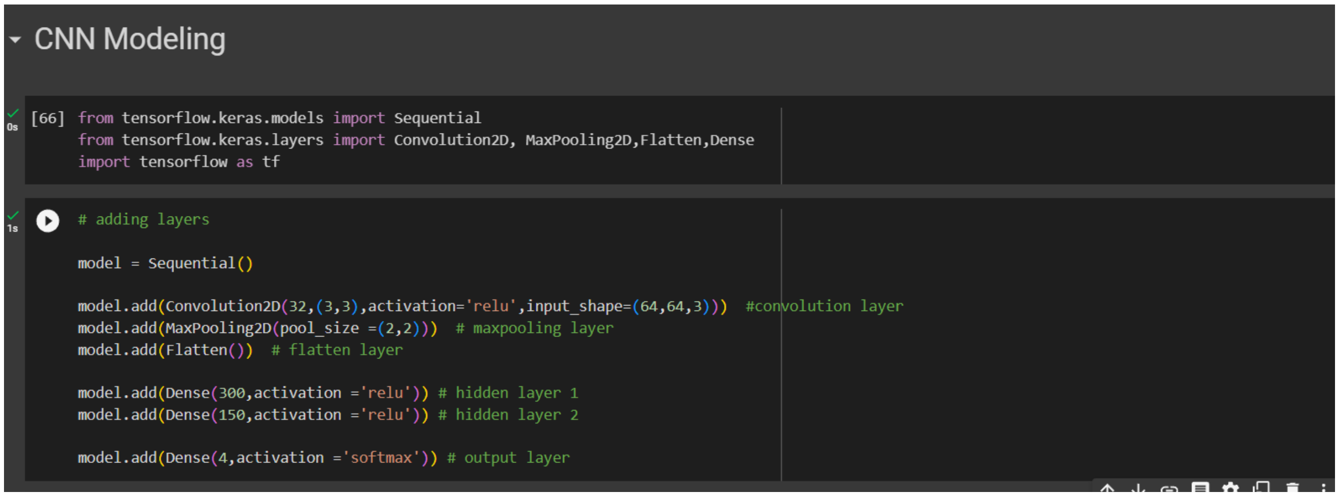
Out of 21165 images, 14818 belongs to training class and 6347 belongs to test class, divided into 4 classes

Step 3: Model Building

Now, we will proceed to construct our model, opting for the pre-trained Xception, a notable convolutional neural network (CNN) architecture renowned for its efficacy in image classification. It is recommended to review the detailed insights on the Xception model provided in the prior knowledge section before embarking on the model-building phase.

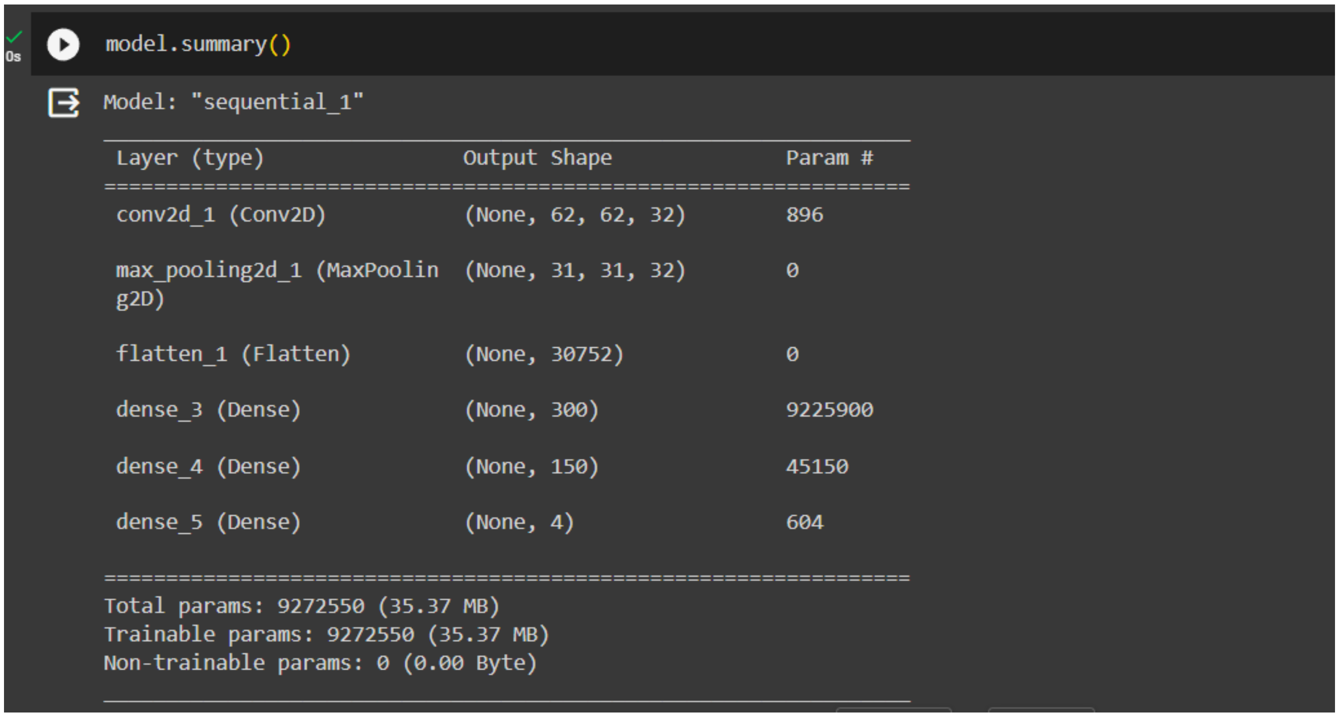
Pre-trained CNN model as a Feature Extractor

In this code snippet, a Convolutional Neural Network (CNN) model is defined using the TensorFlow and Keras libraries. The model architecture consists of a convolutional layer with 32 filters, each of size (3,3), using the rectified linear unit (ReLU) activation function. This is followed by a max-pooling layer to downsample the spatial dimensions. The flattened layer prepares the data for the subsequent dense layers. The model then has two hidden layers with 300 and 150 neurons, respectively, using the ReLU activation function. The final layer is the output layer with 4 neurons, employing the softmax activation function for multi-class classification. The model is compiled with the Adam optimizer, categorical crossentropy loss function, and accuracy as the evaluation metric. This structure represents a basic CNN for image classification tasks with a specified input shape of (64, 64, 3).



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

```
# 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(4,activation ='softmax')) # output layer
```



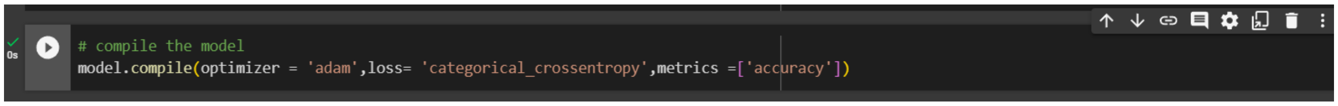
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 62, 62, 32)        896

 max_pooling2d_1 (MaxPoolin  (None, 31, 31, 32)        0
 g2D)

 flatten_1 (Flatten)         (None, 30752)             0

 dense_3 (Dense)             (None, 300)               9225900

 dense_4 (Dense)             (None, 150)               45150

 dense_5 (Dense)             (None, 4)                 604

=================================================================
Total params: 9272550 (35.37 MB)
Trainable params: 9272550 (35.37 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The model.summary() function in Keras provides a concise overview of the neural network architecture, presenting information about the type of layers, their output shapes, and the number of parameters within the model. This summary aids in understanding the structure of the model, including the input and output dimensions, as well as the total number of parameters to be trained. The displayed information facilitates a quick examination of the model's complexity and assists in identifying potential issues or optimizing the network for better performance during training and inference.

Configure the Learning Process

The model.compile function configures the training process for a neural network model in Keras. In this specific instance, the Adam optimizer is chosen as the optimization algorithm, which adapts learning rates for each parameter individually, often leading to faster and more effective convergence during training. The loss function specified is categorical crossentropy, suitable for multi-class classification tasks. This function quantifies the difference between the predicted class probabilities and the actual class labels. Lastly, the 'accuracy' metric is set to monitor the model's performance during training, providing insights into how well the model is correctly classifying instances. This configuration prepares the model for the subsequent training phase, ensuring it is ready to learn from the provided dataset.



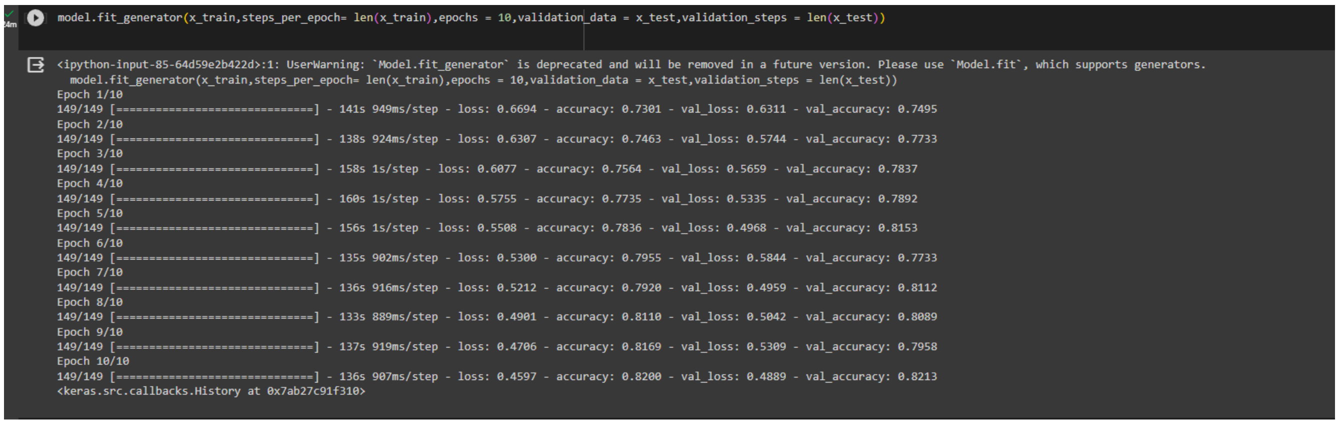
```
# compile the model
model.compile(optimizer = 'adam',loss= 'categorical_crossentropy',metrics =['accuracy'])
```

Train the model

We will now initiate the training of our model with the image dataset. The model is trained over 25 epochs, and after each epoch, the current state of the model is saved if it exhibits the lowest loss encountered thus far. Observing the training process, it is noticeable that the training loss consistently decreases in nearly every epoch up to the 10th epoch, suggesting potential for further model enhancement.
The training process is executed using the fit_generator function, a fundamental tool for training deep learning neural networks.

Arguments:

- **steps_per_epoch:** This parameter defines the total number of steps taken from the generator once one epoch is completed and the subsequent epoch begins. Its value is calculated by dividing the total number of samples in the dataset by the batch size.
- **Epochs:** An integer representing the number of epochs for which we intend to train our model.
- **validation_data:** This can take various forms:
    - A list of inputs and targets.
    - A generator.
    - A list containing inputs, targets, and sample weights. This combination is used to assess the loss and metrics for any model after each epoch.
- **validation_steps:** This parameter is applicable only if the validation_data is a generator. It specifies the total number of steps taken from the generator before it halts at the conclusion of each epoch. Its value is computed by dividing the total number of validation data points in the dataset by the validation batch size.



```
model.fit_generator(x_train,steps_per_epoch= len(x_train),epochs = 10,validation_data = x_test,validation_steps = len(x_test))
```

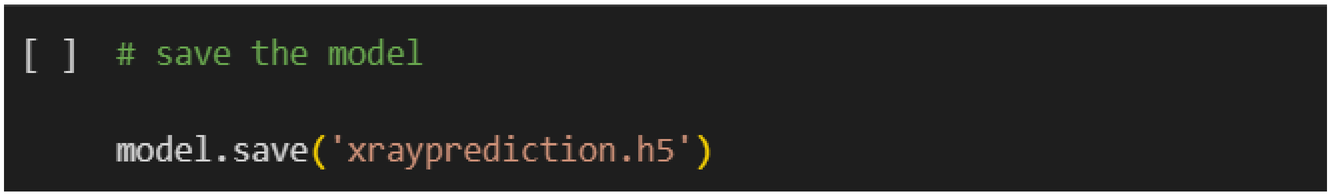
```
<ipython-input-85-64d59e2b422d>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  model.fit_generator(x_train,steps_per_epoch= len(x_train),epochs = 10,validation_data = x_test,validation_steps = len(x_test))
Epoch 1/10
149/149 [==============================] - 141s 949ms/step - loss: 0.6694 - accuracy: 0.7301 - val_loss: 0.6311 - val_accuracy: 0.7495
Epoch 2/10
149/149 [==============================] - 138s 924ms/step - loss: 0.6307 - accuracy: 0.7463 - val_loss: 0.5744 - val_accuracy: 0.7733
Epoch 3/10
149/149 [==============================] - 158s 1s/step - loss: 0.6077 - accuracy: 0.7564 - val_loss: 0.5659 - val_accuracy: 0.7837
Epoch 4/10
149/149 [==============================] - 160s 1s/step - loss: 0.5755 - accuracy: 0.7735 - val_loss: 0.5335 - val_accuracy: 0.7892
Epoch 5/10
149/149 [==============================] - 156s 1s/step - loss: 0.5508 - accuracy: 0.7836 - val_loss: 0.4968 - val_accuracy: 0.8153
Epoch 6/10
149/149 [==============================] - 135s 902ms/step - loss: 0.5300 - accuracy: 0.7955 - val_loss: 0.5844 - val_accuracy: 0.7733
Epoch 7/10
149/149 [==============================] - 136s 916ms/step - loss: 0.5212 - accuracy: 0.7920 - val_loss: 0.4959 - val_accuracy: 0.8112
Epoch 8/10
149/149 [==============================] - 133s 889ms/step - loss: 0.4901 - accuracy: 0.8110 - val_loss: 0.5042 - val_accuracy: 0.8089
Epoch 9/10
149/149 [==============================] - 137s 919ms/step - loss: 0.4706 - accuracy: 0.8169 - val_loss: 0.5309 - val_accuracy: 0.7958
Epoch 10/10
149/149 [==============================] - 136s 907ms/step - loss: 0.4597 - accuracy: 0.8200 - val_loss: 0.4889 - val_accuracy: 0.8213
<keras.src.callbacks.History at 0x7ab27c91f310>
```

From the above run time, we can easily observe that at 10th epoch the model is givingthe better accuracy of 0.8200.

Step 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.

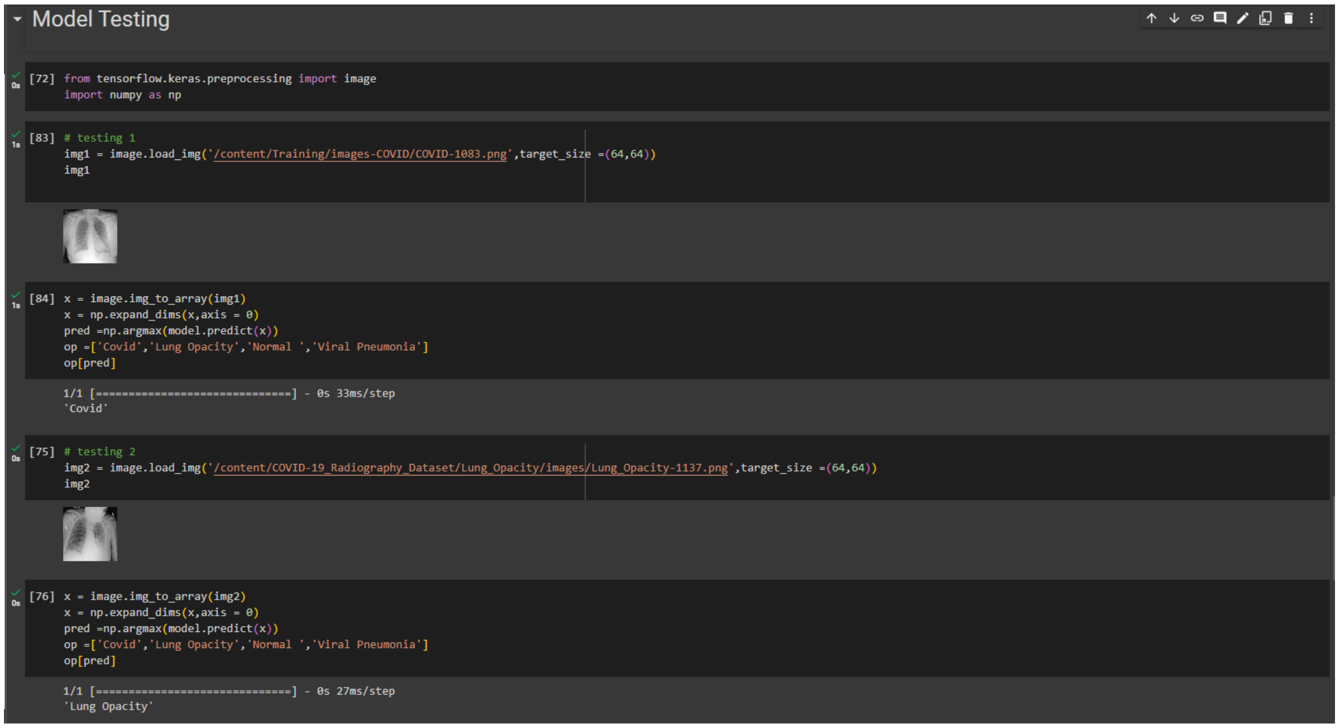


```
[ ]  # save the model

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

Step 5: Testing the Model

Testing a model involves assessing its performance on a dataset that it hasn't been exposed to during training. This phase is pivotal in the development of any machine learning model, providing insights into its ability to generalize effectively to new and unseen data.

There are five test cases testing all of the four prediction classes.



Model Testing

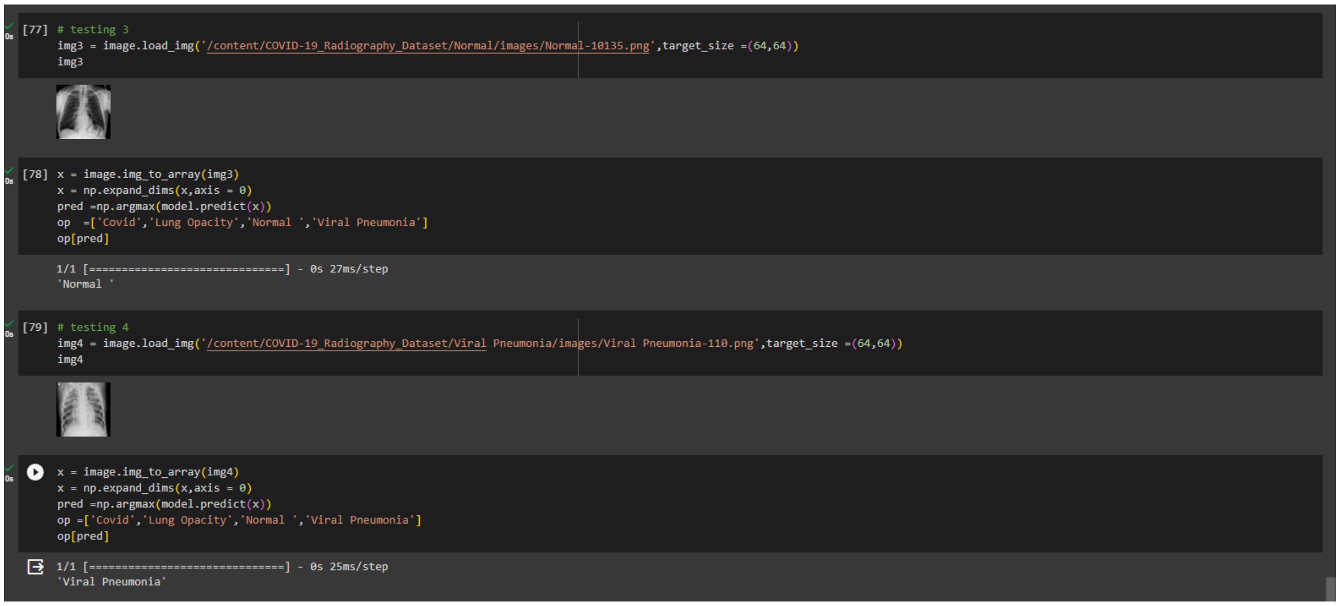
```
[72] from tensorflow.keras.preprocessing import image
     import numpy as np
```

```
[83] # testing 1
     img1 = image.load_img('/content/Training/images-COVID/COVID-1083.png',target_size =(64,64))
     img1
```

```
[84] x = image.img_to_array(img1)
     x = np.expand_dims(x,axis = 0)
     pred =np.argmax(model.predict(x))
     op =['Covid','Lung Opacity','Normal ','Viral Pneumonia']
     op[pred]

     1/1 [==============================] - 0s 33ms/step
     'Covid'
```

```
[75] # testing 2
     img2 = image.load_img('/content/COVID-19_Radiography_Dataset/Lung_Opacity/images/Lung_Opacity-1137.png',target_size =(64,64))
     img2
```

```
[76] x = image.img_to_array(img2)
     x = np.expand_dims(x,axis = 0)
     pred =np.argmax(model.predict(x))
     op =['Covid','Lung Opacity','Normal ','Viral Pneumonia']
     op[pred]

     1/1 [==============================] - 0s 27ms/step
     'Lung Opacity'
```



```
[77] # testing 3
     img3 = image.load_img('/content/COVID-19_Radiography_Dataset/Normal/images/Normal-10135.png',target_size =(64,64))
     img3
```

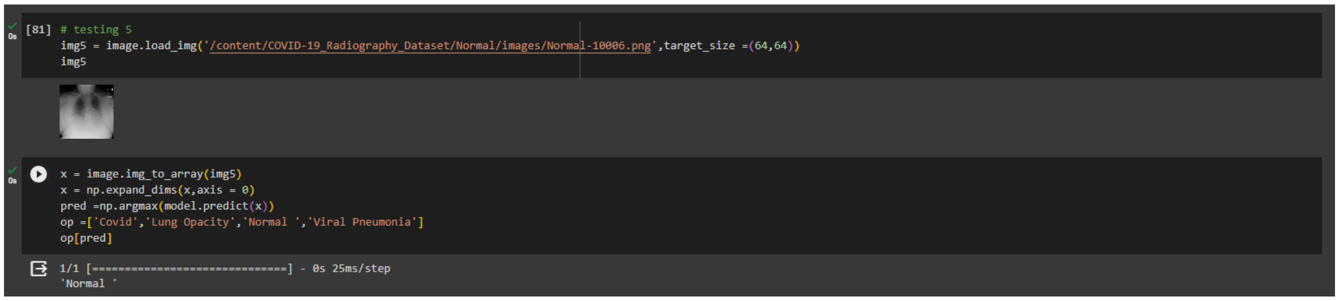
```
[78] x = image.img_to_array(img3)
     x = np.expand_dims(x,axis = 0)
     pred =np.argmax(model.predict(x))
     op  =['Covid','Lung Opacity','Normal ','Viral Pneumonia']
     op[pred]
```

```
1/1 [==============================] - 0s 27ms/step
'Normal '
```

```
[79] # testing 4
     img4 = image.load_img('/content/COVID-19_Radiography_Dataset/Viral Pneumonia/images/Viral Pneumonia-110.png',target_size =(64,64))
     img4
```

```
x = image.img_to_array(img4)
x = np.expand_dims(x,axis = 0)
pred =np.argmax(model.predict(x))
op =['Covid','Lung Opacity','Normal ','Viral Pneumonia']
op[pred]
```

```
1/1 [==============================] - 0s 25ms/step
'Viral Pneumonia'
```



```
[81] # testing 5
     img5 = image.load_img('/content/COVID-19_Radiography_Dataset/Normal/images/Normal-10006.png',target_size =(64,64))
     img5
```

```
x = image.img_to_array(img5)
x = np.expand_dims(x,axis = 0)
pred =np.argmax(model.predict(x))
op =['Covid','Lung Opacity','Normal ','Viral Pneumonia']
op[pred]
```

```
1/1 [==============================] - 0s 25ms/step
'Normal '
```

Step 6: Application Building

This section involves the development of a web application seamlessly integrated with the previously constructed model. A user-friendly interface is presented, allowing users to input values for predictions. These input values are then passed to the saved model, and the resulting predictions are displayed on the user interface. The tasks in this section include:

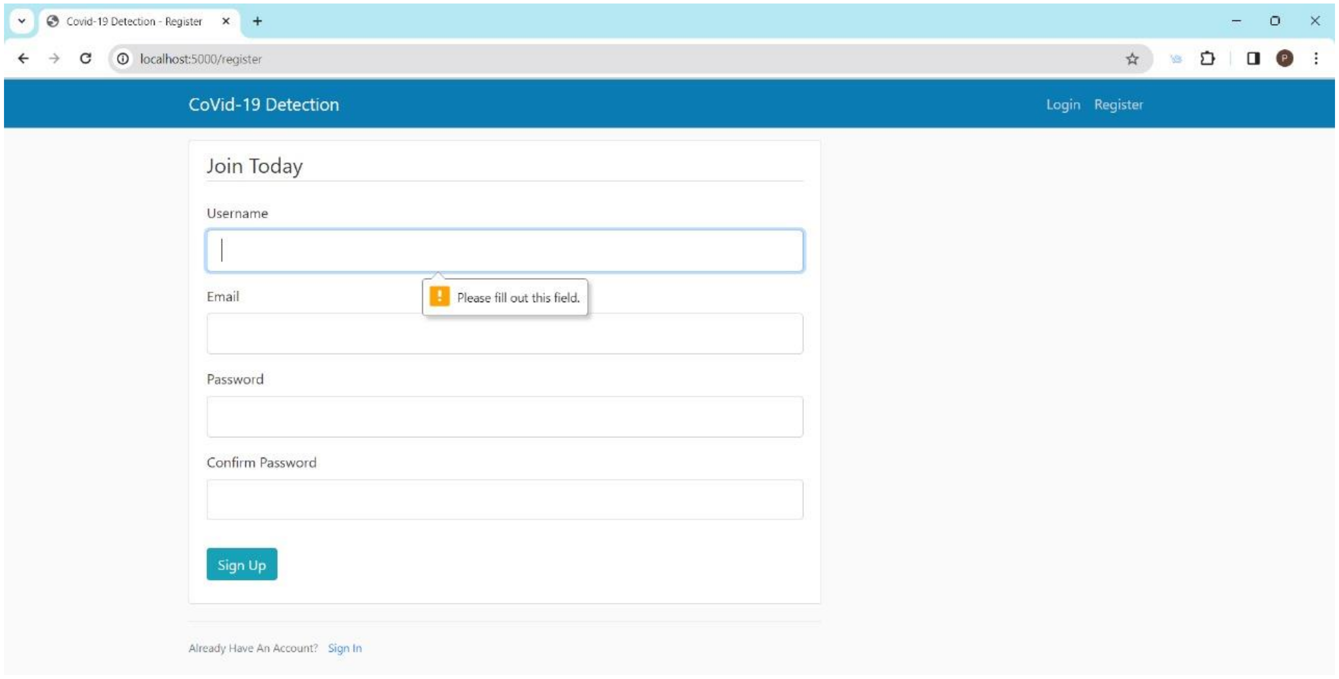
- Building HTML Pages
- Building server side script

Building Html Pages:

For this project create one HTML file namely

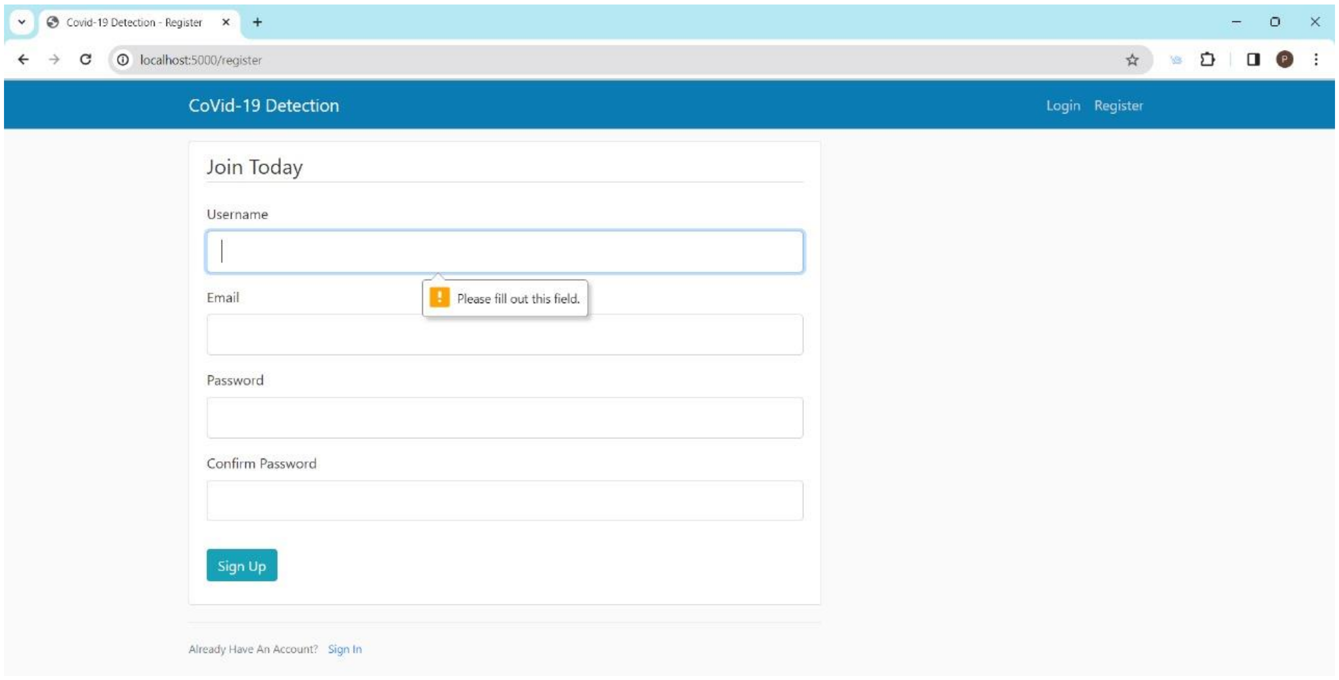
- Registration.html

In Registration page it asks for username, email id, and password.



- login.html

The login page asks for the Email ID and password that you have registered with.


Covid-19 Detection - Register

localhost:5000/register

CoVid-19 Detection

Login Register

Join Today

Username

Please fill out this field.

Email

Password

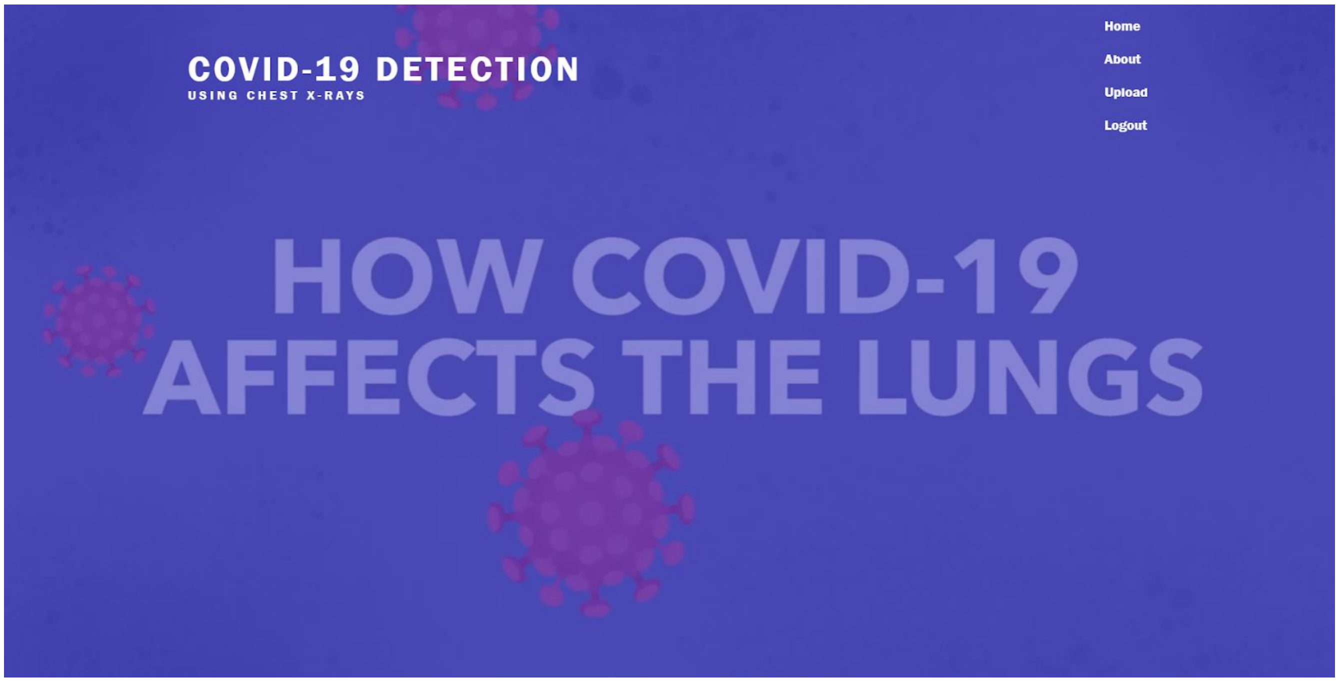
Confirm Password

Sign Up

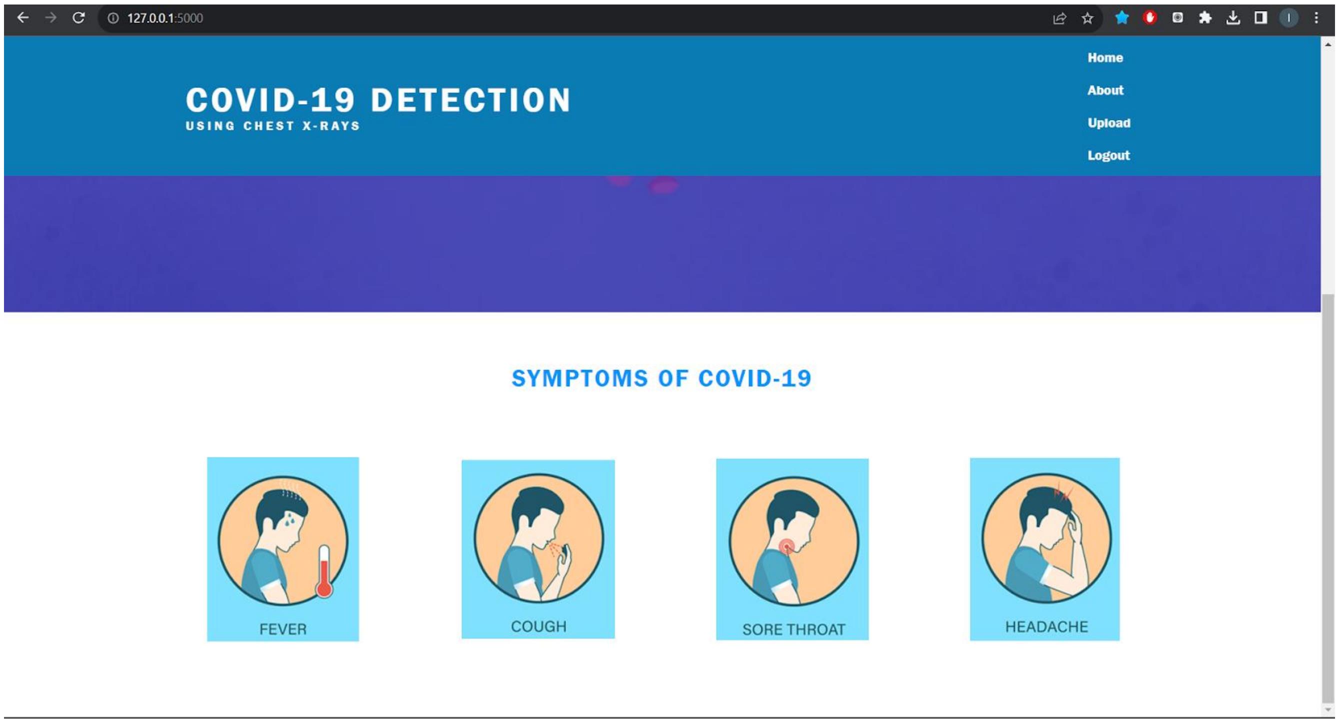
Already Have An Account? Sign In

- Index.html

The index page(Home page) contains a informative video which shows how covid-19 infects our lungs and contains 4 buttons namely Home which is the index page, About, Upload and logout .Which will redirect accordingly

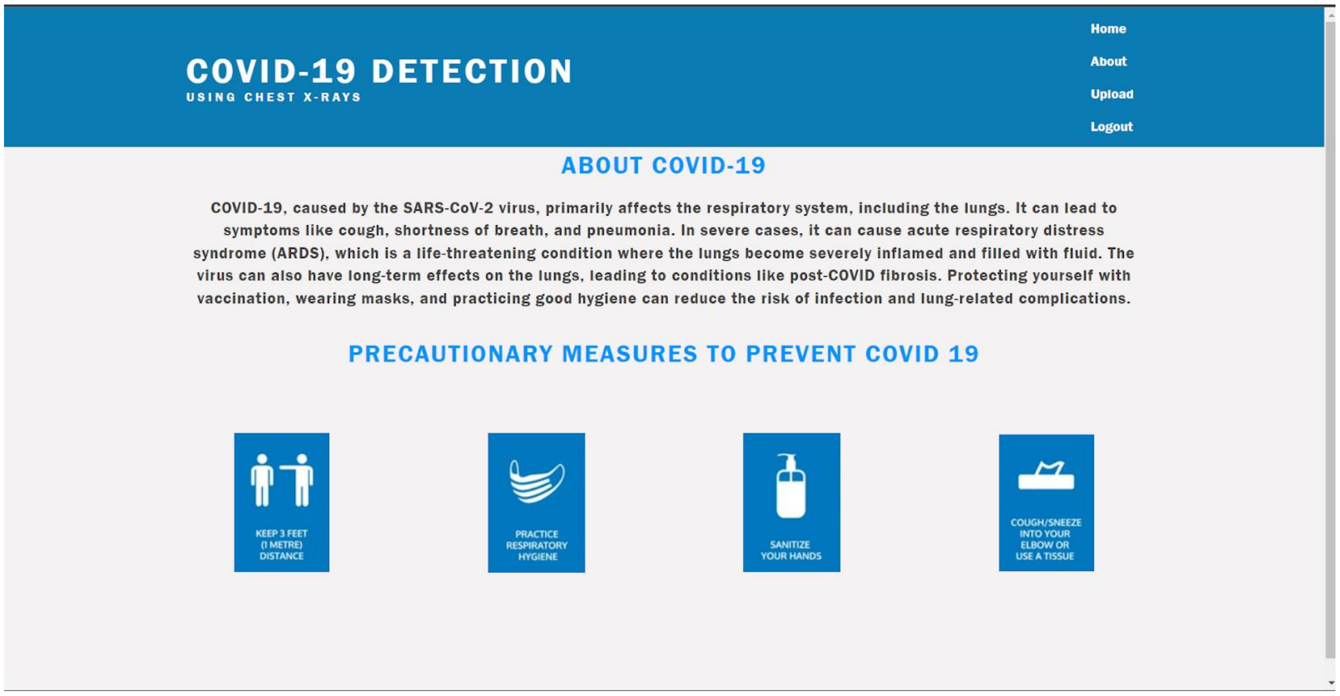


Below the video we have a few symptoms of Covid-19



- About.html

When we click about it will redirected to a new page which information about covid-19 and few precautionary measures to prevent covid-19.



ABOUT COVID-19

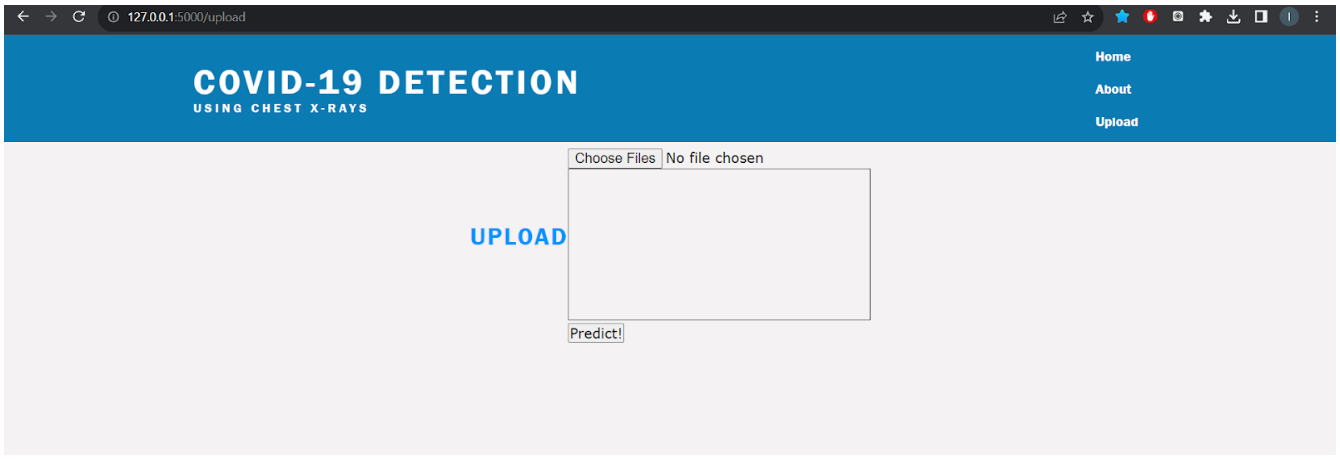
COVID-19, caused by the SARS-CoV-2 virus, primarily affects the respiratory system, including the lungs. It can lead to symptoms like cough, shortness of breath, and pneumonia. In severe cases, it can cause acute respiratory distress syndrome (ARDS), which is a life-threatening condition where the lungs become severely inflamed and filled with fluid. The virus can also have long-term effects on the lungs, leading to conditions like post-COVID fibrosis. Protecting yourself with vaccination, wearing masks, and practicing good hygiene can reduce the risk of infection and lung-related complications.

PRECAUTIONARY MEASURES TO PREVENT COVID 19

KEEP 3 FEET (1 METRE) DISTANCE
PRACTICE RESPIRATORY HYGIENE
SANITIZE YOUR HANDS
COUGH/SNEEZE INTO YOUR ELBOW OR USE A TISSUE

- Upload.html

Finally, we have our upload page where the user can upload the picture and get the desired result



Build Python code:

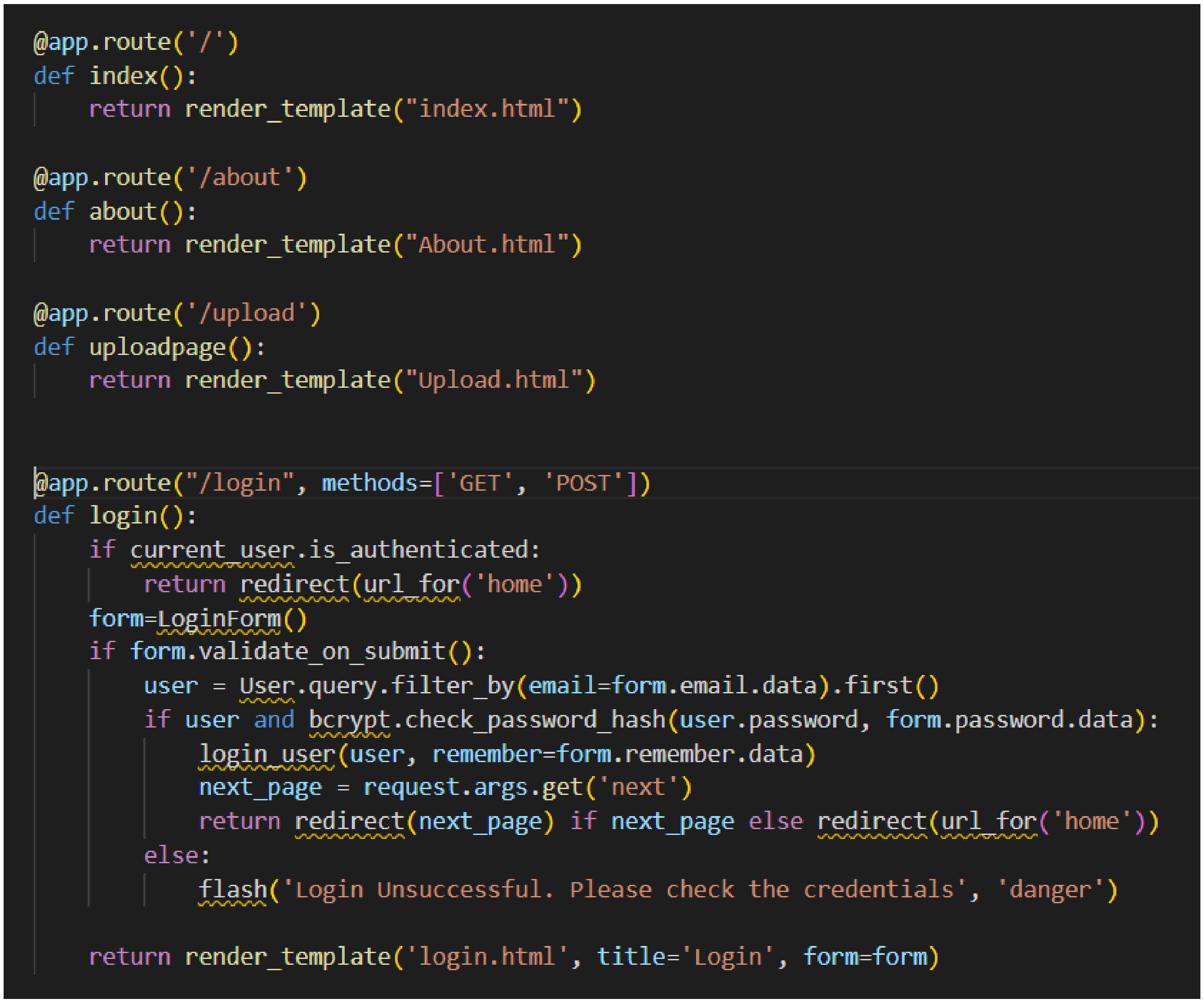
Import the libraries

```
from tensorflow import keras
from flask import Flask, render_template, request
import os
import numpy as np

from datetime import datetime
from detection import db, login_manager
from flask_login import UserMixin
```

Loading the saved model and initializing the flask app

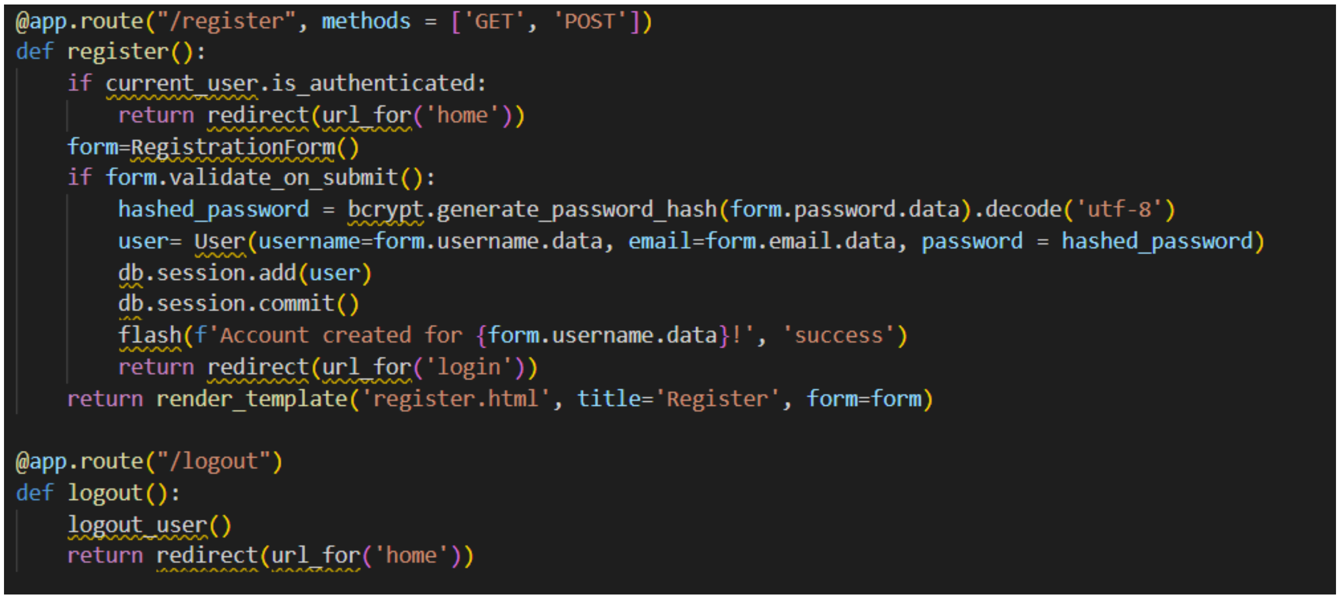
```
app = Flask(__name__)
model = keras.models.load_model(r"xrayprediction.h5", compile=False)
```

Render HTML pages:



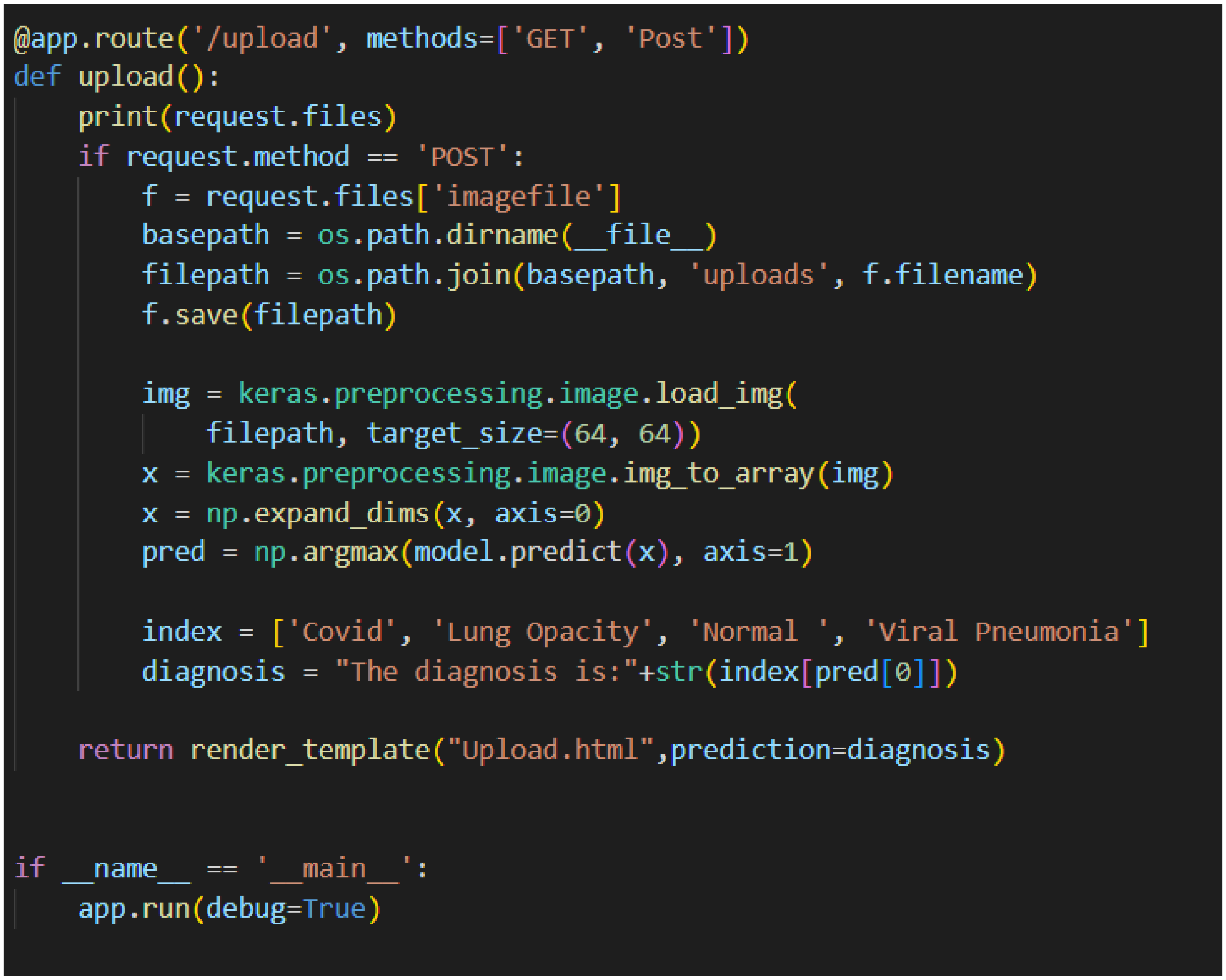
```
@app.route('/')
def index():
    return render_template("index.html")

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

@app.route('/upload')
def uploadpage():
    return render_template("Upload.html")


@app.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form=LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user, remember=form.remember.data)
            next_page = request.args.get('next')
            return redirect(next_page) if next_page else redirect(url_for('home'))
        else:
            flash('Login Unsuccessful. Please check the credentials', 'danger')

    return render_template('login.html', title='Login', form=form)
```

This code snippet represents a Flask web application with different routes. The '/' route renders the 'index.html' template, while the '/about' and '/upload' routes render 'About.html' and 'Upload.html' templates, respectively. The '/login' route is associated with a login form and employs both GET and POST methods. If a user is already authenticated, they are redirected to the home page; otherwise, the login form is displayed. Upon form submission, the user's credentials are verified, and if valid, they are logged in. In case of unsuccessful login attempts, a flash message is displayed. This structure facilitates navigation between different sections of the web application, handles user authentication, and integrates a login form for secure access.



```python
@app.route("/register", methods = ['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form=RegistrationForm()
    if form.validate_on_submit():
        hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user= User(username=form.username.data, email=form.email.data, password = hashed_password)
        db.session.add(user)
        db.session.commit()
        flash(f'Account created for {form.username.data}!', 'success')
        return redirect(url_for('login'))
    return render_template('register.html', title='Register', form=form)

@app.route("/logout")
def logout():
    logout_user()
    return redirect(url_for('home'))
```

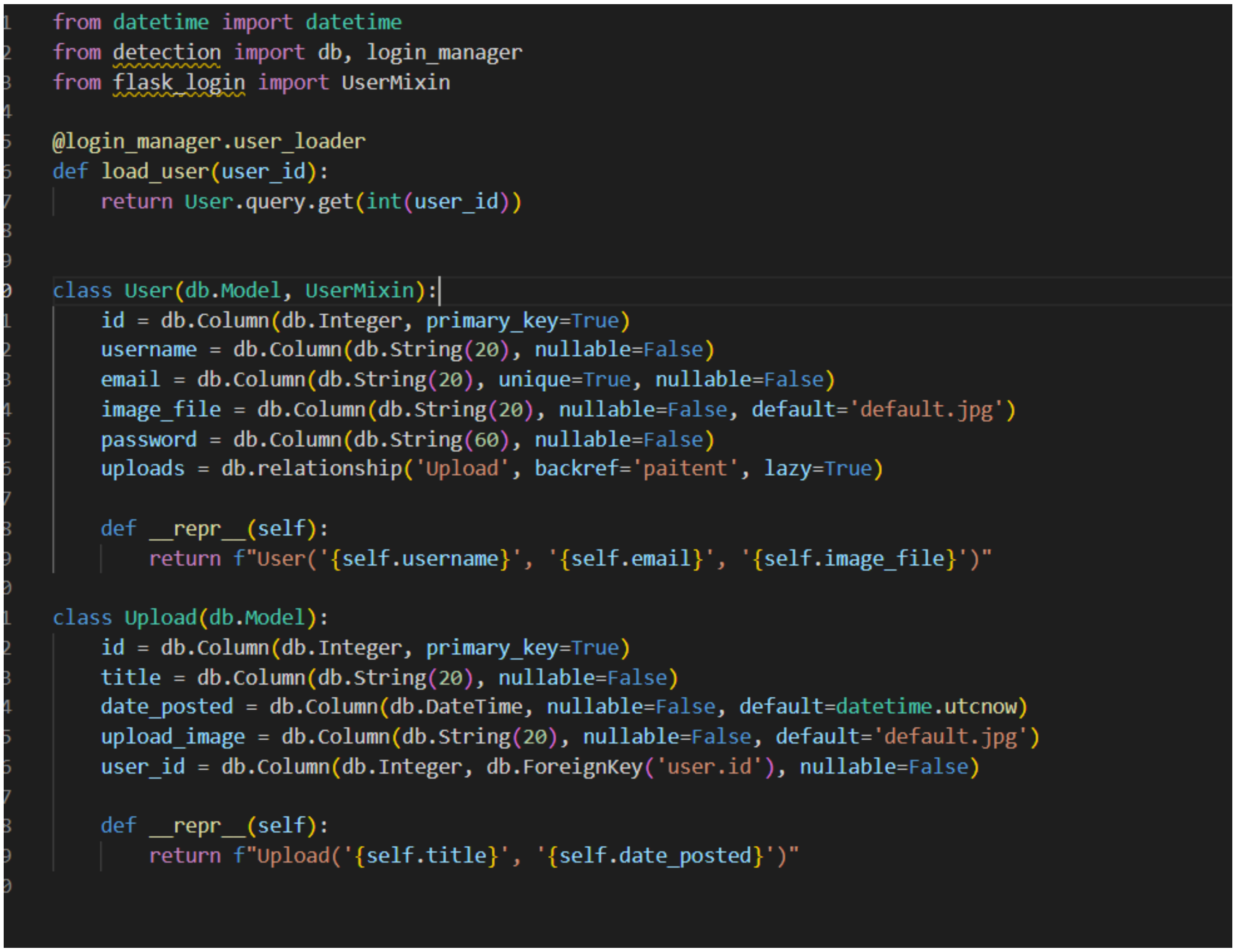
This Flask web application snippet defines two routes: "/register" and "/logout". The "/register" route, associated with both GET and POST methods, presents a registration form to users. If a user is already authenticated, they are redirected to the home page; otherwise, the registration form is displayed. Upon successful form submission, the user's password is hashed, and a new user is created in the database. A flash message confirms the account creation, and the user is redirected to the login page. The "/logout" route logs out the current user and redirects them to the home page. These routes collectively handle user registration, authentication, and logout functionalities in the web application.



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

        img = keras.preprocessing.image.load_img(
            filepath, target_size=(64, 64))
        x = keras.preprocessing.image.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        pred = np.argmax(model.predict(x), axis=1)

        index = ['Covid', 'Lung Opacity', 'Normal ', 'Viral Pneumonia']
        diagnosis = "The diagnosis is:"+str(index[pred[0]])

    return render_template("Upload.html",prediction=diagnosis)


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

This Flask application route, '/upload', handles both GET and POST requests. Upon a POST request, it retrieves the uploaded image file and saves it to the 'uploads' directory. The image is then preprocessed, resized to (64, 64), and converted into a format suitable for model prediction. The trained model predicts the class of the image, and the corresponding diagnosis is determined using an index mapping. The result is displayed in the 'Upload.html' template. This route allows users to upload an image, receive a diagnosis based on the trained model, and view the prediction on the web interface. The application is run in debug mode when executed directly.

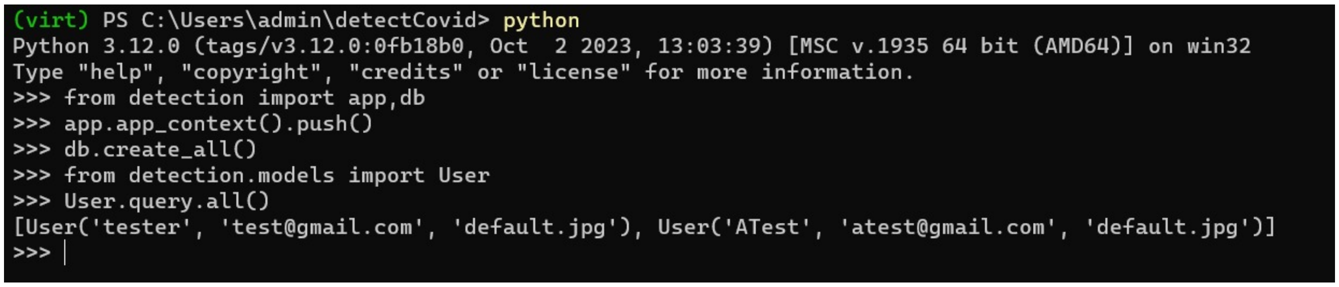
Database Connectivity


```python
from datetime import datetime
from detection import db, login_manager
from flask_login import UserMixin

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))


class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), nullable=False)
    email = db.Column(db.String(20), unique=True, nullable=False)
    image_file = db.Column(db.String(20), nullable=False, default='default.jpg')
    password = db.Column(db.String(60), nullable=False)
    uploads = db.relationship('Upload', backref='paitent', lazy=True)

    def __repr__(self):
        return f"User('{self.username}', '{self.email}', '{self.image_file}')"

class Upload(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(20), nullable=False)
    date_posted = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    upload_image = db.Column(db.String(20), nullable=False, default='default.jpg')
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

    def __repr__(self):
        return f"Upload('{self.title}', '{self.date_posted}')"
```

This code segment defines two classes, User and Upload, which represent the data models in a Flask web application. The User class inherits from both db.Model and UserMixin (from Flask-Login), incorporating essential attributes like user ID, username, email, profile image file, password hash, and a one-to-many relationship with the `Upload` class. The Upload class, also a subclass of db.Model, contains attributes for upload information, including title, posting date, uploaded image file, and a foreign key linking it to a specific user. These classes are integral components of the application's database schema, facilitating the storage and retrieval of user and upload data. The __repr__ method in each class defines a string representation for instances, aiding in debugging and logging.

Run the application

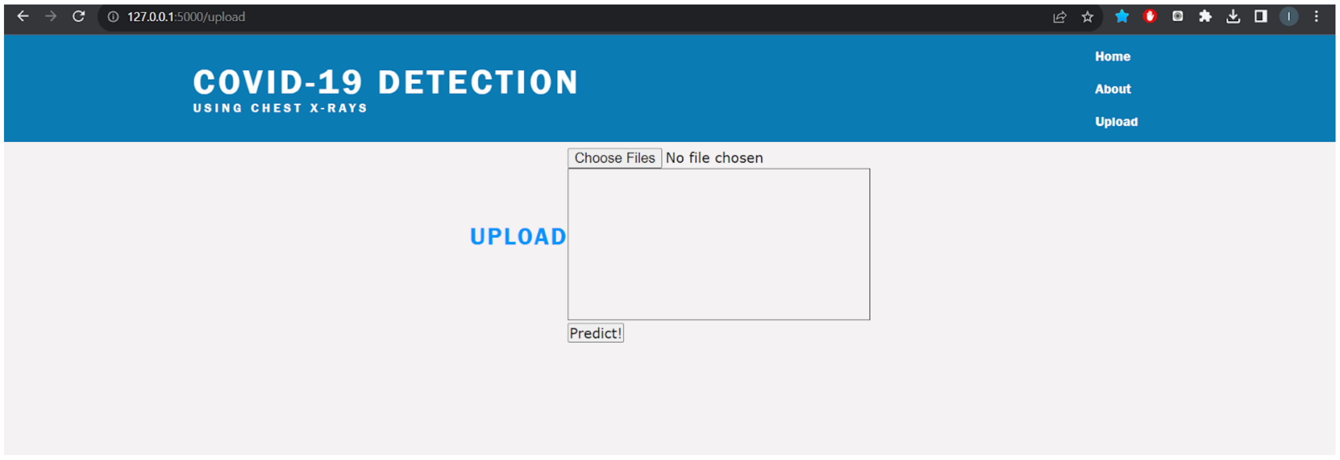
- Open the Command prompt from the start menu.
- Type python and press Enter.

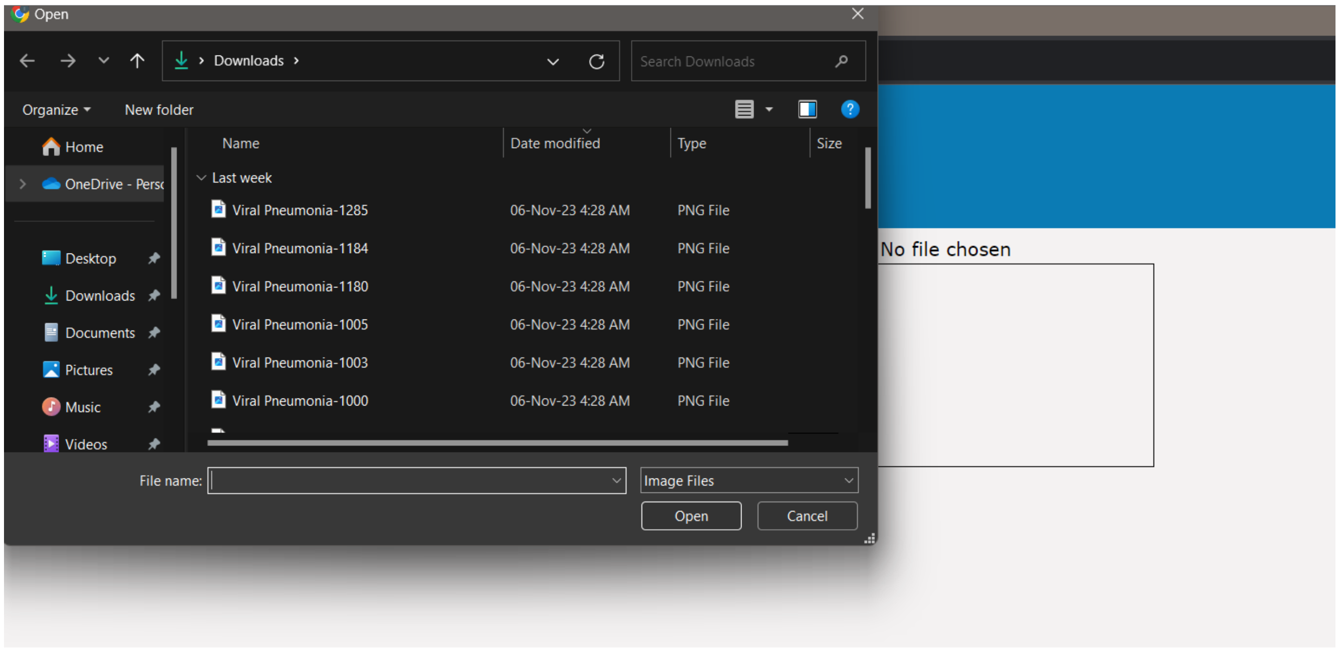


```
(virt) PS C:\Users\admin\detectCovid> python
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from detection import app,db
>>> app.app_context().push()
>>> db.create_all()
>>> from detection.models import User
>>> User.query.all()
[User('tester', 'test@gmail.com', 'default.jpg'), User('ATest', 'atest@gmail.com', 'default.jpg')]
>>> 
```

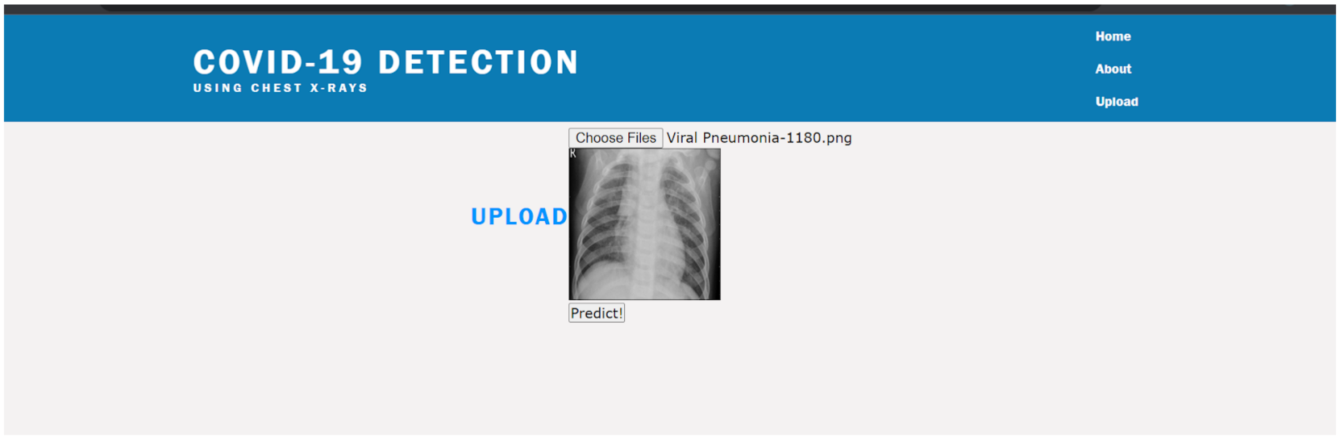
Uploading in website

Input 1:





Name	Date modified	Type	Size
Last week			
Viral Pneumonia-1285	06-Nov-23 4:28 AM	PNG File	
Viral Pneumonia-1184	06-Nov-23 4:28 AM	PNG File	
Viral Pneumonia-1180	06-Nov-23 4:28 AM	PNG File	
Viral Pneumonia-1005	06-Nov-23 4:28 AM	PNG File	
Viral Pneumonia-1003	06-Nov-23 4:28 AM	PNG File	
Viral Pneumonia-1000	06-Nov-23 4:28 AM	PNG File	



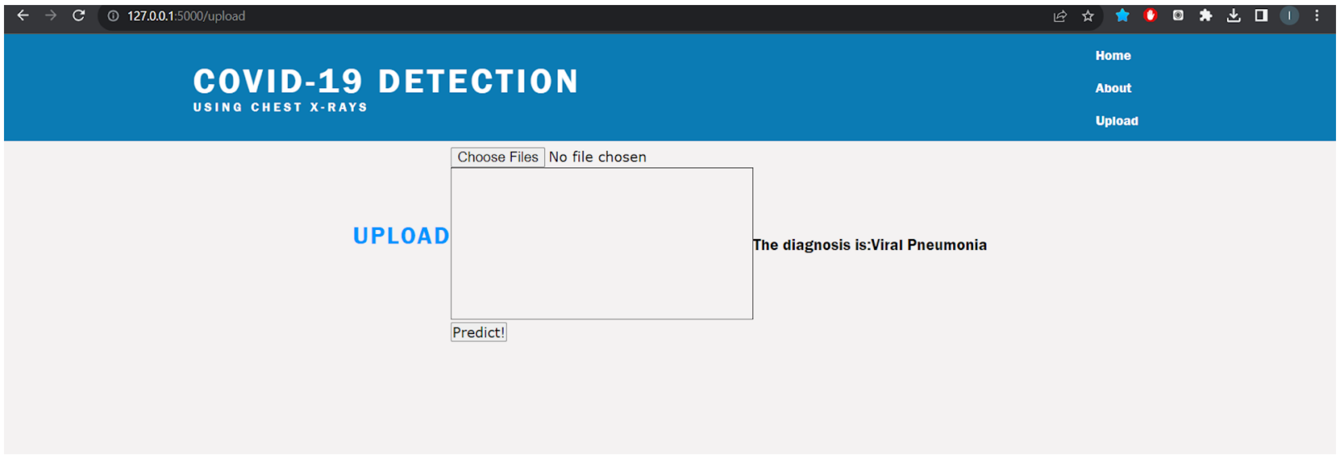
COVID-19 DETECTION

USING CHEST X-RAYS

Viral Pneumonia-1180.png

Click predict

Output:1



127.0.0.1:5000/upload

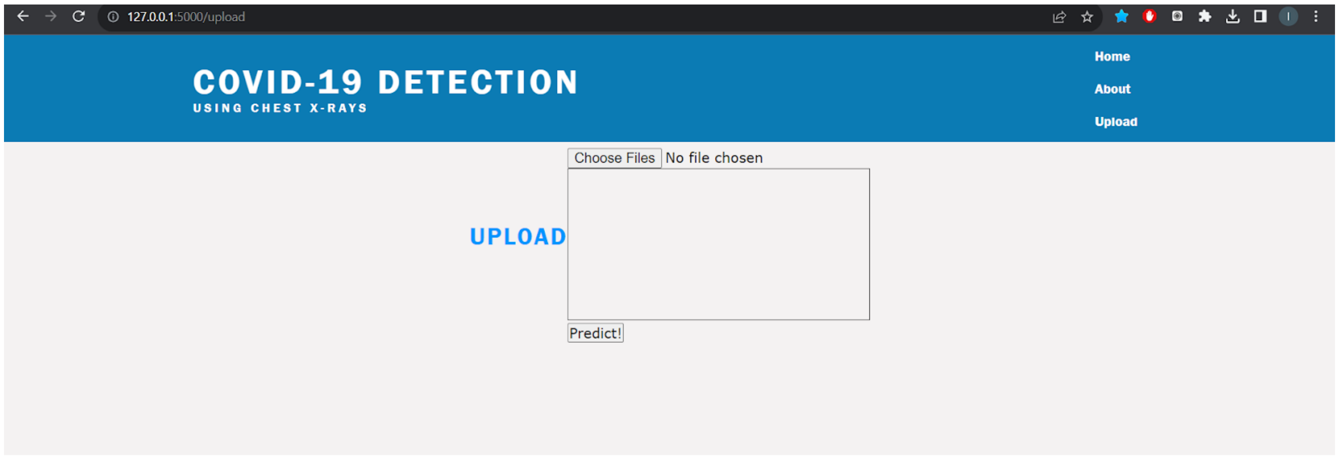
COVID-19 DETECTION

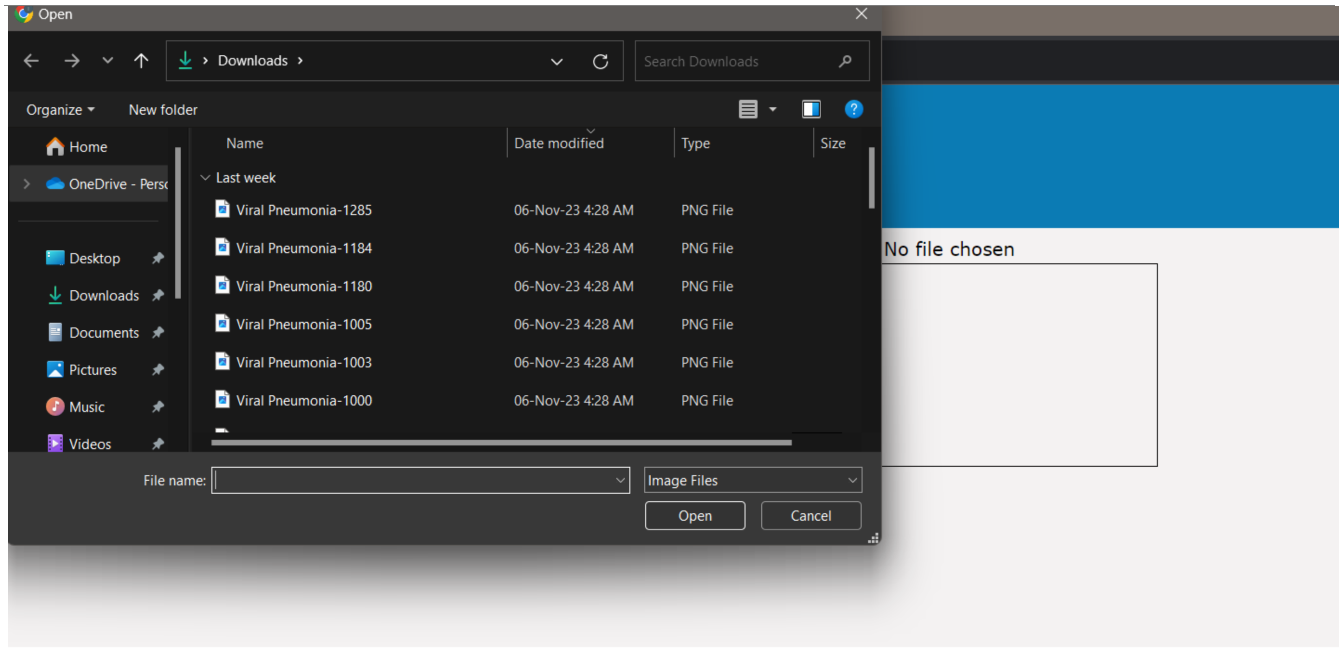
USING CHEST X-RAYS

No file chosen

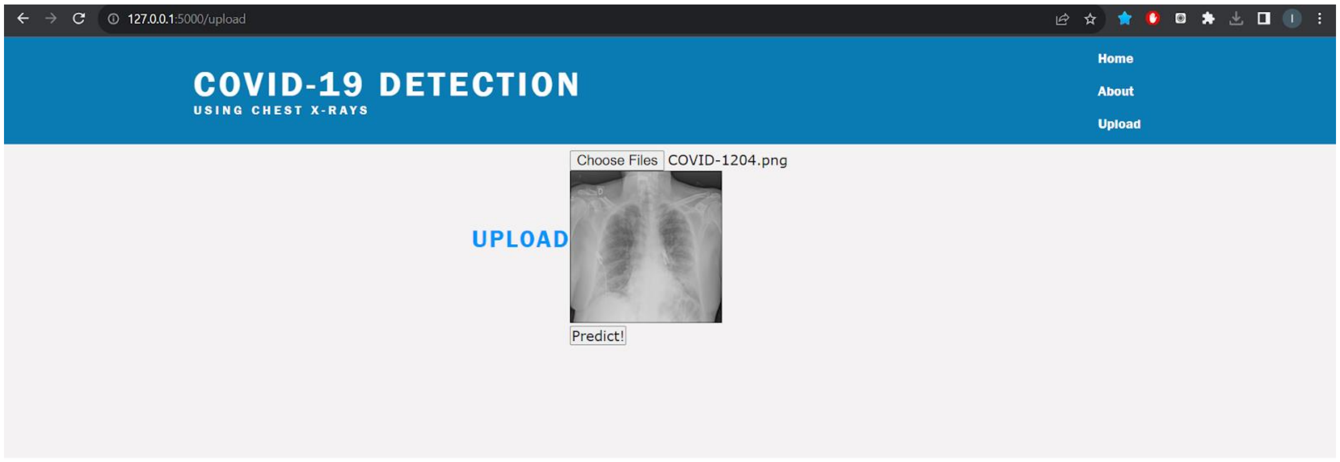
The diagnosis is:Viral Pneumonia

Input:2

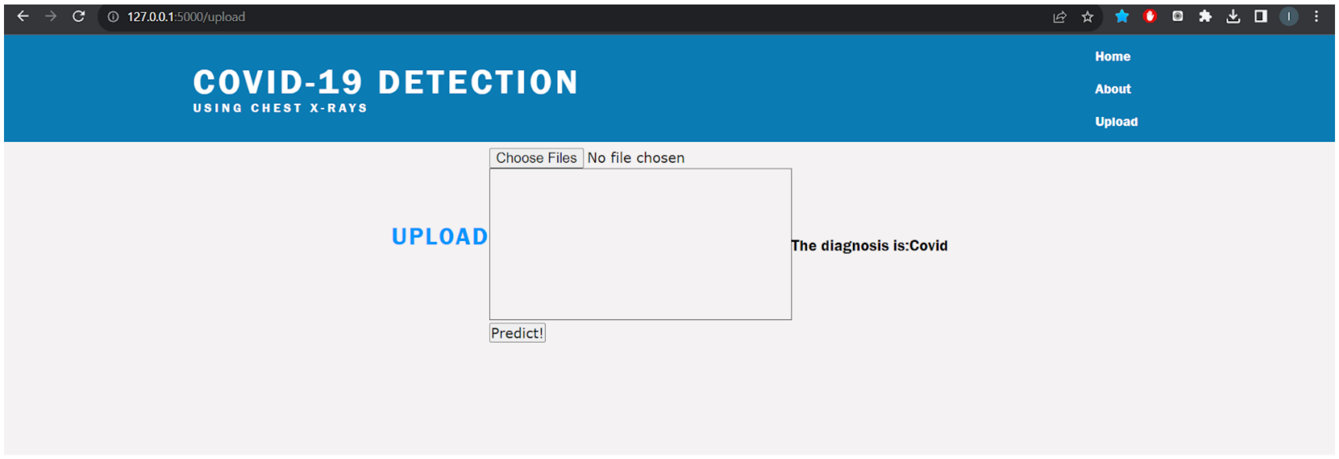




Name	Date modified	Type	Size
Last week			
Viral Pneumonia-1285	06-Nov-23 4:28 AM	PNG File	
Viral Pneumonia-1184	06-Nov-23 4:28 AM	PNG File	
Viral Pneumonia-1180	06-Nov-23 4:28 AM	PNG File	
Viral Pneumonia-1005	06-Nov-23 4:28 AM	PNG File	
Viral Pneumonia-1003	06-Nov-23 4:28 AM	PNG File	
Viral Pneumonia-1000	06-Nov-23 4:28 AM	PNG File	



Output: 2



The diagnosis is:Covid