

7. CASE STUDY

Development of a Web-Based X-ray Image Disease Classification System

Problem Statement:

In medical diagnostics, accurate X-ray classification is vital for effective treatment. Current manual methods can lead to errors and delays. This project aims to develop a CNN-based system to automate chest X-ray classification, focusing on distinguishing pneumonia from normal lungs, to enhance diagnostic accuracy and efficiency

Solution Overview:

This procedure outlines the steps to build an Android application for smart retail using TensorFlow Lite for object detection on product shelves. Note that this is a high-level guide, and each step might involve additional sub-tasks depending on your specific needs.

Data Collection and Preprocessing:

- Image Dataset: Take high-resolution pictures of store shelves with various products, ensuring different lighting conditions and arrangements. Include close-ups for small items and barcodes. [\[5\]](#)

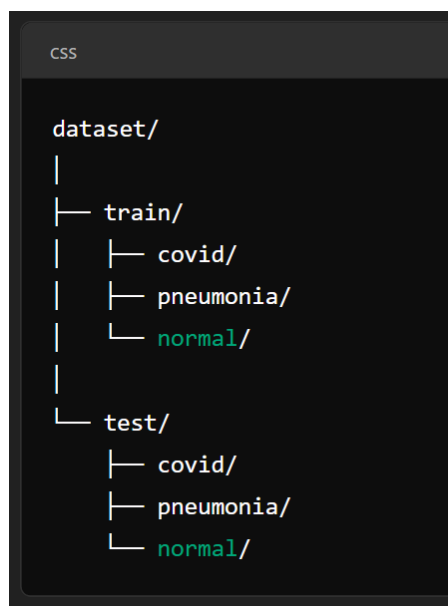
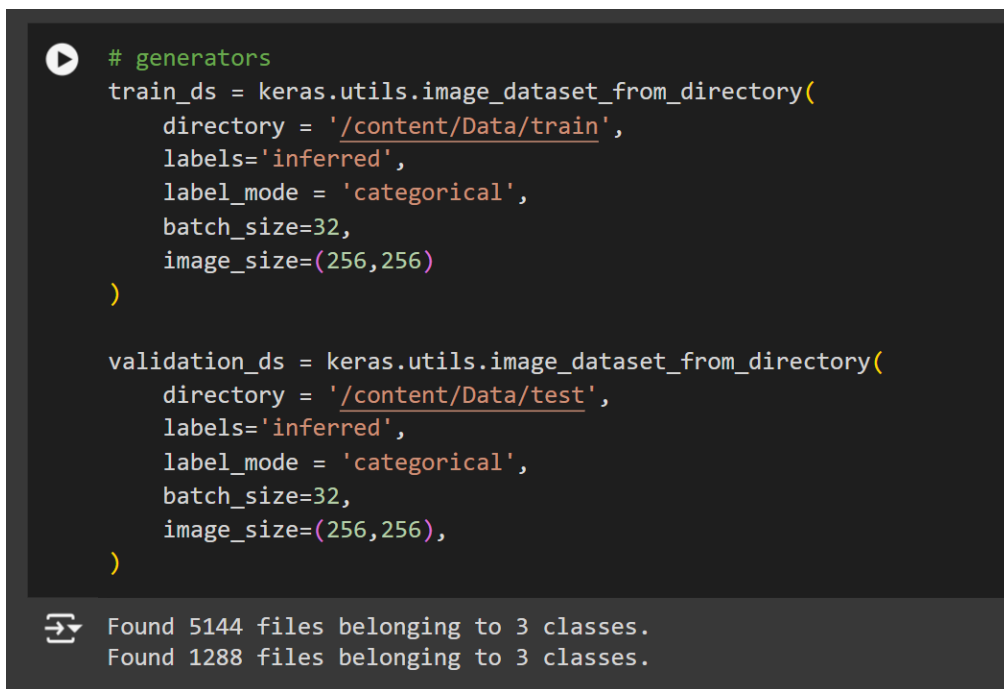


Fig 7.1 represents the directory of the taken dataset

- Data Splitting: Divide labeled data into training, validation, and test sets. Use the training set to train the model, the validation set to fine-tune hyperparameters, and the test set to evaluate final model performance.
 - Training dataset images = 5144
 - Testing dataset images = 1288
 - Total images in dataset = 6432
 - Total classes = 3



```
# generators
train_ds = keras.utils.image_dataset_from_directory(
    directory = '/content/Data/train',
    labels='inferred',
    label_mode = 'categorical',
    batch_size=32,
    image_size=(256,256)
)

validation_ds = keras.utils.image_dataset_from_directory(
    directory = '/content/Data/test',
    labels='inferred',
    label_mode = 'categorical',
    batch_size=32,
    image_size=(256,256),
)

Found 5144 files belonging to 3 classes.
Found 1288 files belonging to 3 classes.
```

Fig 7.2 represents generators used for training and validation

- Preprocessing:
 - Resize images to a consistent size required by the model.
 - Normalize pixel values, typically converting to a range of 0-1.
 - Apply Normalization and Dropout techniques to increase data variety and improve model generalizability.

This version captures the essential steps of your data collection and preprocessing process in a clear and concise manner.

```

# Normalize
def process(image,label):
    image = tf.cast(image/255. ,tf.float32)
    return image,label

train_ds = train_ds.map(process)
validation_ds = validation_ds.map(process)

```

Fig 7.3 represents the preprocessing of image in dataset

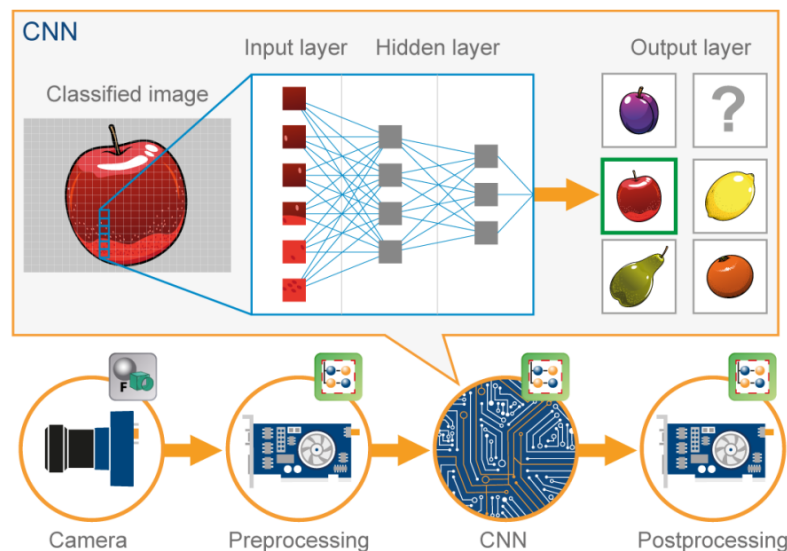


Fig 7.4 represents the working of CNN model

Model Selection and Training (Using Python):

- Objective: Develop a robust image classification model to accurately categorize images (e.g., X-ray images of different conditions).
- Data Preparation: Collect and preprocess data, including resizing, normalization.
- Model Architecture: Choose and implement a Convolutional Neural Network (CNN) or other suitable architecture for the classification task.

The CNN model begins by importing essential modules from TensorFlow's Keras API. These include the Sequential model, which allows for the construction of a linear stack of layers, and various layer types such as convolutional, batch normalization, max pooling, flatten, dense, and dropout layers.

- A sequential model is initialized, setting up a container where layers will be added sequentially.

```
python Copy code  
  
model = Sequential()
```

- The first layer added is a 2D convolutional layer with 32 filters and a 3x3 kernel size. This layer uses 'valid' padding, meaning no padding is added around the input, and ReLU activation, which introduces non-linearity to the model. The input shape is specified as 256x256 pixels with 3 color channels, defining the dimensions of the input images.

```
python Copy code  
  
model.add(Conv2D(32, kernel_size=(3,3), padding='valid', activation='relu', input_shape=(2
```

ext, a batch normalization layer is added to normalize the activations of the previous convolutional layer, which helps speed up training and improves model stability.

```
python Copy code  
  
model.add(BatchNormalization())
```

- A max pooling layer is then added with a 2x2 pool size and a stride of 2. This layer reduces the spatial dimensions of the feature map by selecting the maximum value in each 2x2 window, effectively downsampling the input.

```
python Copy code  
  
model.add(MaxPooling2D(pool_size=(2,2), strides=2, padding='valid'))
```

- The process is repeated with a second convolutional layer, this time with 64 filters, followed by another batch normalization layer. This increases the complexity of the features extracted from the images.

```
python Copy code  
  
model.add(Conv2D(64, kernel_size=(3,3), padding='valid', activation='relu'))
```

- Another max pooling layer is added to further reduce the spatial dimensions.
- A third convolutional layer with 128 filters is added, followed by batch normalization to handle more complex features.

```
python Copy code  
  
model.add(Conv2D(128, kernel_size=(3,3), padding='valid', activation='relu'))
```

- A third max pooling layer is added to further downsample the feature map.
- The output from the convolutional layers is then flattened into a one-dimensional array, preparing it for the dense layers that follow.

```
python Copy code  
  
model.add(Flatten())
```

- The first dense (fully connected) layer with 128 units and ReLU activation is added, allowing the model to learn more complex representations.

```
python Copy code  
  
model.add(Dense(128, activation='relu'))
```

- To prevent overfitting, a dropout layer with a rate of 0.1 is added, which randomly sets 10% of the input units to zero during training.

```
python Copy code  
  
model.add(Dropout(0.1))
```

- A second dense layer with 64 units and ReLU activation is added for further representation learning, followed by another dropout layer with a rate of 0.1 for regularization.

```
python Copy code  
  
model.add(Dense(64, activation='relu'))
```

- Finally, the output layer is a dense layer with 3 units and a softmax activation function, indicating that the model is designed for a three-class classification problem. The softmax function ensures that the output is a probability distribution over the three classes.

```
python Copy code  
  
model.add(Dense(3, activation='softmax'))
```

- Training Process: Train the model using labeled data, adjusting hyperparameters to optimize performance. Monitor metrics such as accuracy and loss to evaluate the training progress.

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

- Optimizer : Adam(Adaptive Moment Estimation)
 - Loss : Categorical (Multi class Label)
 - Metrics : Accuracy (Classification)
- Fitting the model: The code trains the neural network model using the 'fit' method, where 'train_ds' is the training dataset and 'validation_ds' is the validation dataset. The training process runs for one epoch, meaning the model will see each training sample once. The 'fit' method also tracks the training and validation performance, which is stored in the 'history' object for later analysis.

```
[ ] history = model.fit(train_ds,epochs=10,validation_data=validation_ds)
```

- Evaluation Metrics:

- Training accuracy = 0.96 and Testing accuracy = 0.93
- Train loss = 0.0872 and Test loss = 0.2576
- Training Precision = 0.96 and Test Precision = 0.93
- Train Recall score = 0.96 and Test Recall score = 0.93

```

# Check if the model training completed successfully.
# If it did, the history object should have recorded the metrics.
if history.history:
    print('Train accuracy:',history.history['accuracy'][-1]*100)
    print('Test accuracy:',history.history['val_accuracy'][-1]*100)
    print('Train loss:',history.history['loss'][-1])
    print('Test loss:',history.history['val_loss'][-1])
    print('Precision:',history.history['precision'][-1])
    print('Recall:',history.history['recall'][-1])
    print('Train precision:',history.history['precision'][-1])
    print('Test precision:',history.history['val_precision'][-1])
    print('Train recall:',history.history['recall'][-1])
    print('Test recall:',history.history['val_recall'][-1])
    print("Model training completed successfully.")
else:
    print("Model training did not complete successfully. Check for errors during training.")

```

```

Train accuracy: 96.92845940589905
Test accuracy: 93.0900633351135
Train loss: 0.08726757019758224
Test loss: 0.2576131522655487
Precision: 0.9694730639457703
Recall: 0.9692845940589905
Train precision: 0.9694730639457703
Test precision: 0.9316239356994629
Train recall: 0.9692845940589905
Test recall: 0.930900633351135
Model training completed successfully.

```

Fig 7.5 represents the evaluated metrics of model

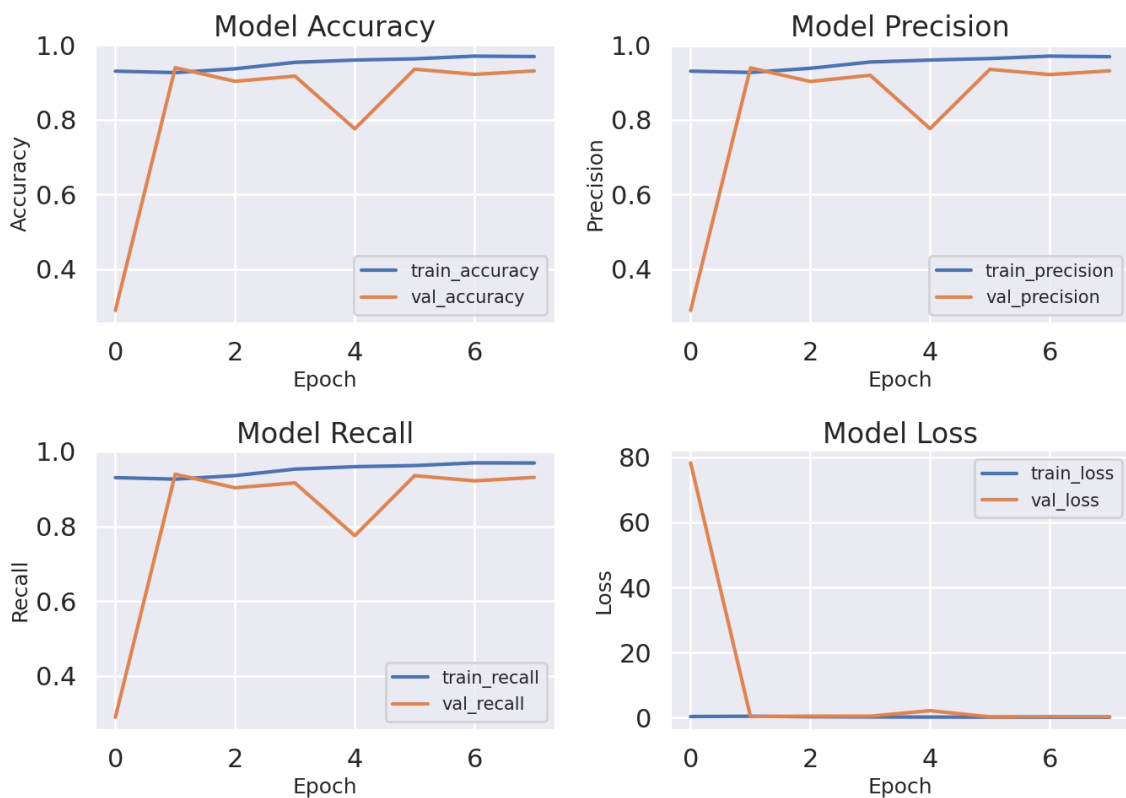


Fig 7.6 represents plot of metrics during epochs

Model Conversion and Optimization Using NGrok:

- **Model Conversion:** Convert the trained model into TensorFlow Lite format to enable deployment on mobile devices. Use the TensorFlow Lite Converter to create a `.tflite` file from the trained model.
- **Optimization Techniques:** Apply quantization to reduce model size and enhance performance. This step may involve converting the model to use integer values instead of floating-point numbers.
- **Using NGrok:** Leverage NGrok to expose local development environments for remote testing and integration. NGrok provides secure tunnels to your local server, enabling you to access and test the model conversion and optimization process remotely.

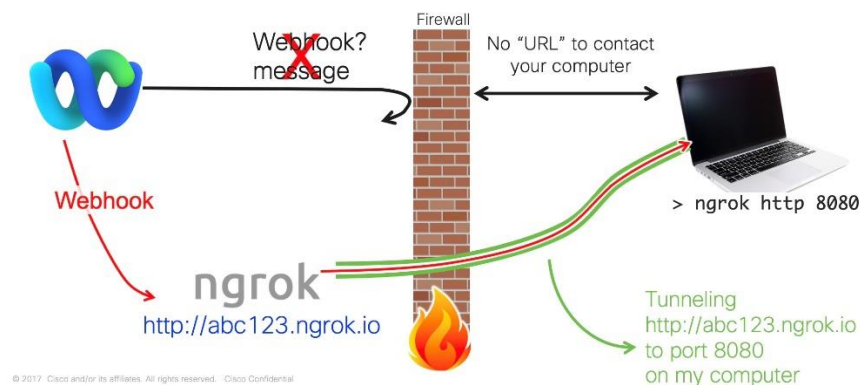


Fig 7.7 represents the working of ngrok website

Website using Ngrok:

- **Start Your Local Server:** Run your local web server (e.g., Flask or Node.js) on a specified port.
- **Run NGrok:** Execute ``ngrok http <port>`` in your terminal to create a public URL for your local server.
- **Access and Share:** Use the NGrok-provided public URL to remotely access and test your application, sharing it with others as needed.
- **Security:** Monitor access and secure sensitive data, shutting down NGrok when testing is complete.

Image Preprocess part:

```
from flask import Flask, request, jsonify, render_template
from werkzeug.utils import secure_filename
import os
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import cv2
import base64
from pyngrok import ngrok

app = Flask(__name__)
model = load_model('/content/drive/MyDrive/XrayDetection/pathmodel3.keras')

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def preprocess_image(img_path, target_size=(256, 256)):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array / 255.0
    return img_array

def classify_image(img_path, model):
    input_image = preprocess_image(img_path)
    predictions = model.predict(input_image)
    predicted_class_index = np.argmax(predictions, axis=1)
    class_names = ['COVID19', 'NORMAL', 'PNEUMONIA']
    predicted_class_label = class_names[predicted_class_index[0]]
    return predicted_class_label, predictions[0]
```

Fig 7.8 represents the preprocess part of Website

Backend part:

```
@app.route('/upload', methods=['POST'])
def upload_file():
    global model

    if 'file' not in request.files:
        return jsonify({'result': "No file part"})

    file = request.files['file']

    if file.filename == '':
        return jsonify({'result': "No selected file"})

    if file and allowed_file(file.filename):
        try:
            # Save the file temporarily
            filename = secure_filename(file.filename)
            file_path = os.path.join('/tmp', filename)
            file.save(file_path)

            # Classify the image
            predicted_class_label, probabilities = classify_image(file_path, model)

            # Load the original image for display
            original_img = cv2.imread(file_path)
            original_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2RGB)

            # Remove the temporary file
            os.remove(file_path)

            # Prepare data to send to the template
            img_str = cv2.imencode('.jpg', original_img)[1].tobytes()
            img_base64 = "data:image/jpeg;base64," + base64.b64encode(img_str).decode('utf-8')
```

Fig 7.9 represents the Backend part of website

Website received from ngrok:

```
Ngrok Tunnel: NgrokTunnel: "https://e1a7-35-185-123-106.ngrok-free.app" -> "http://localhost:5000"
* Serving Flask app '_main_'
* Debug mode: off
INFO:werkzeug: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://172.28.0.12:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug:127.0.0.1 - - [16/Jul/2024 13:17:51] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/Jul/2024 13:17:51] "GET /favicon.ico HTTP/1.1" 404 -
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7a1598dd5b40> triggered tf.function retr
1/1 [=====] - 0s 171ms/step
INFO:werkzeug:127.0.0.1 - - [16/Jul/2024 13:18:02] "POST /upload HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/Jul/2024 13:18:16] "GET / HTTP/1.1" 200 -
1/1 [=====] - 0s 72ms/step
INFO:werkzeug:127.0.0.1 - - [16/Jul/2024 13:18:27] "POST /upload HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/Jul/2024 13:18:47] "GET / HTTP/1.1" 200 -
1/1 [=====] - 0s 66ms/step
```

Fig 7.10 represents the website received from ngrok

Data Visualization and Integration:

- **Data Visualization:** Create visual representations of model performance and data insights using tools like Matplotlib, Seaborn, or Plotly. Include visualizations such as accuracy plots, confusion matrices, and loss curves.
- **Integration into Application:** Embed visualization components into your web or mobile application to display real-time insights and analytics. Ensure the visualizations are user-friendly and provide valuable information to users.
- **Data Integration:** Connect and integrate data from various sources into your application and model. Develop data pipelines for smooth data handling and ensure that your application can process and update data in real-time.

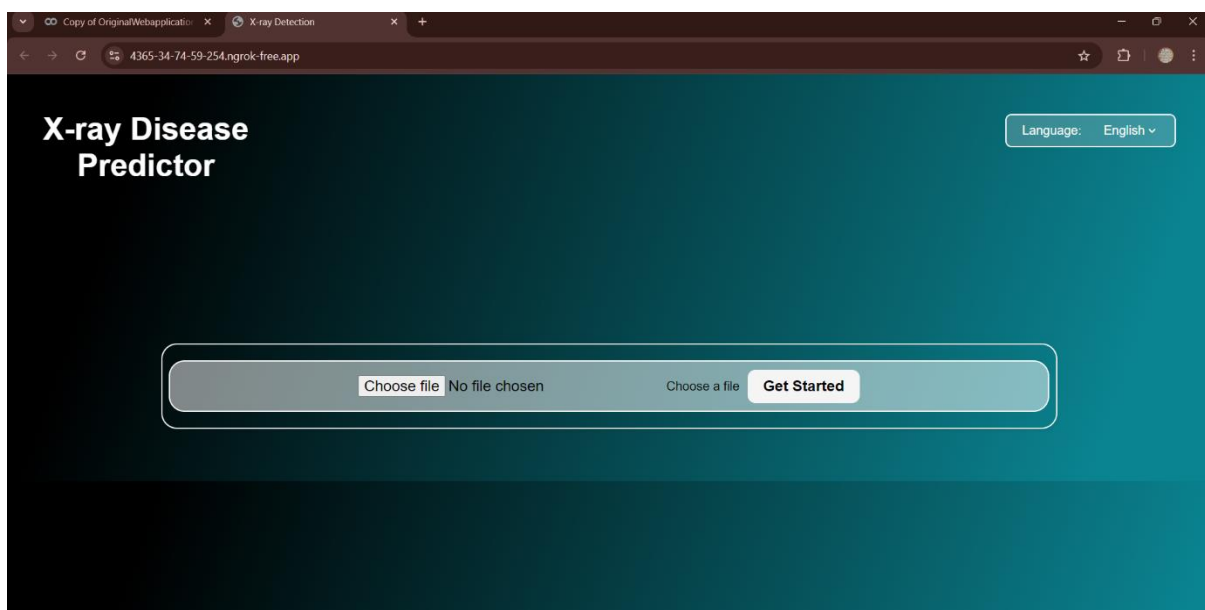


Fig 7.11 represents the web interface of X-ray Disease Predicator

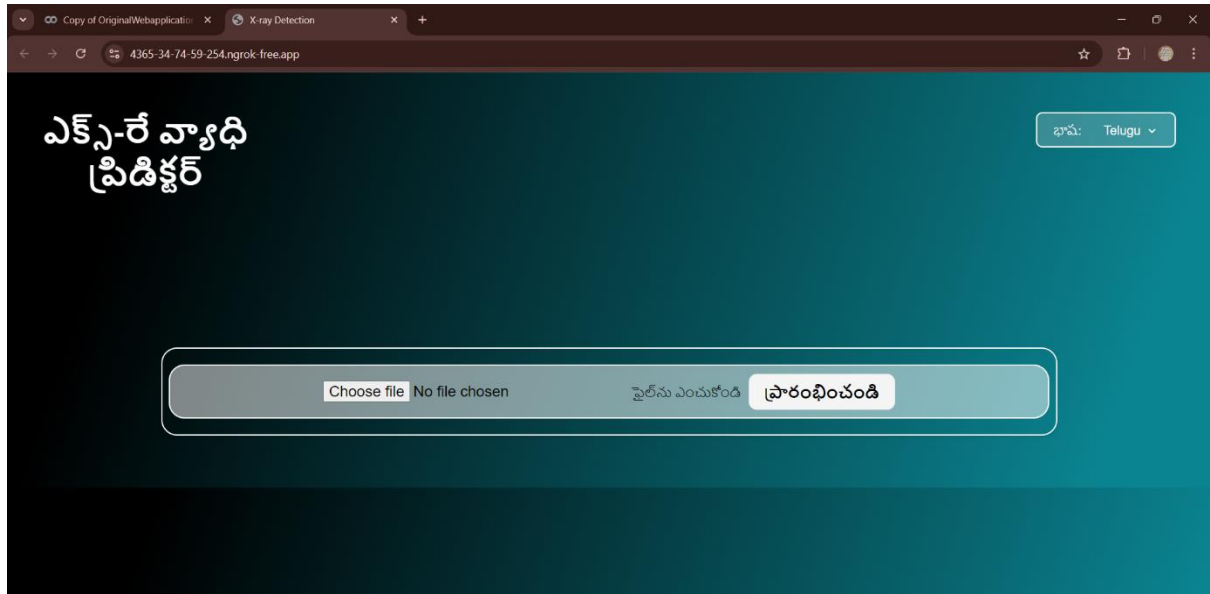


Fig 7.12 represents that the website is in three languages(English,Hindi,Telugu)

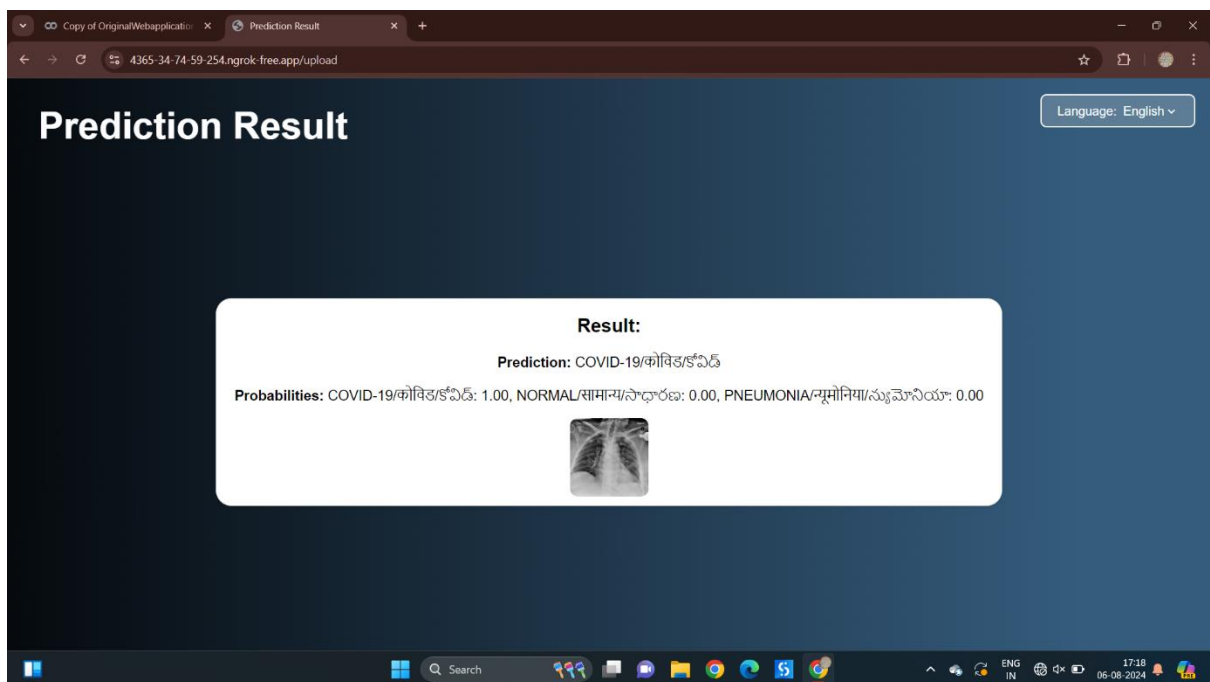


Fig 7.13 represents the Prediction result of Covid X-ray Image

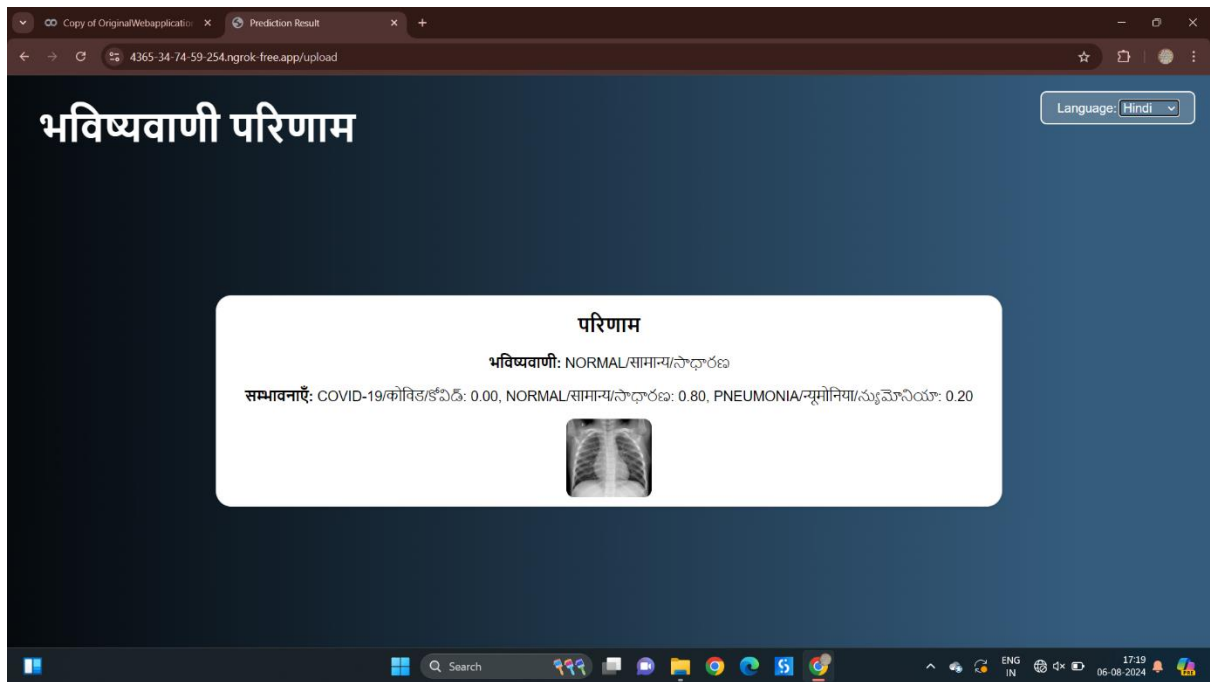


Fig 7.14 represents the Prediction result of Normal X-ray Image

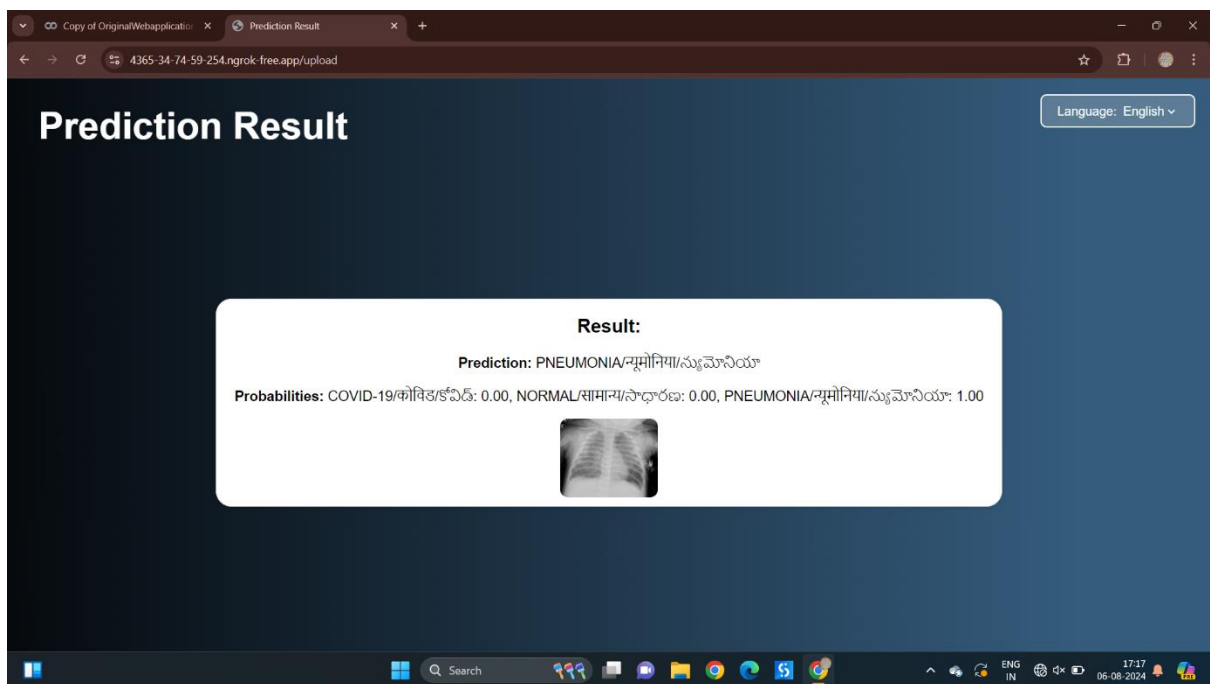
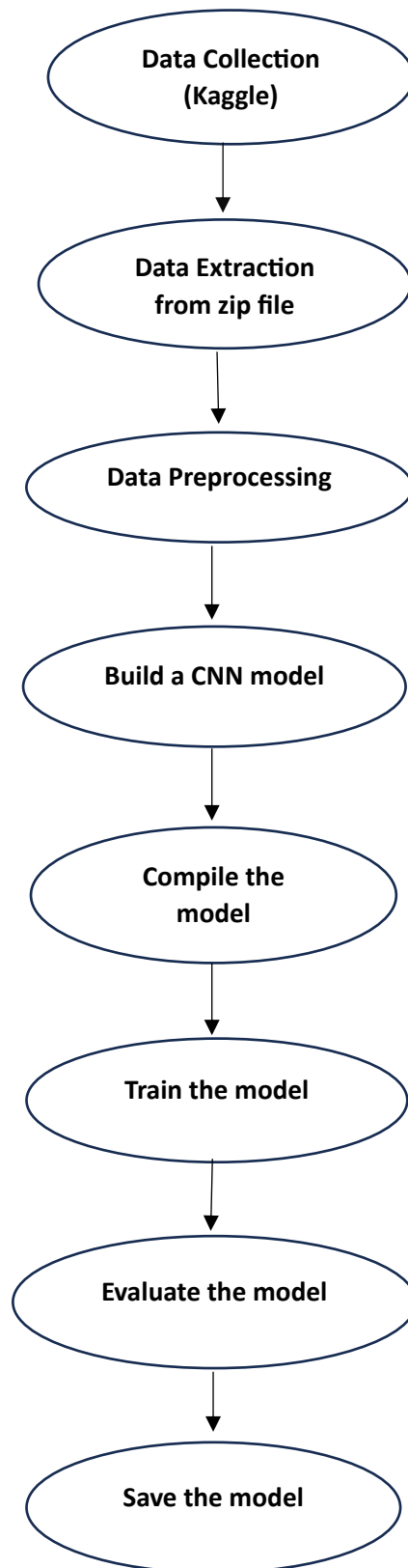


Fig 7.15 represents the Prediction result of Pneumonia X-ray Image

FLOW CHART



Continue.....

