Model Development Phase

Project Name: COVID-19 Chest X-Ray Image Classification

Model Selection Report

In this project, multiple deep learning approaches were evaluated for classifying chest X-ray images into three categories: COVID-19, Bacterial Pneumonia, and Normal.
Factors such as training accuracy, validation accuracy, confusion matrix, and loss curves were considered to select the most suitable model.

Models:

Model 1: VGG16 (Transfer Learning) :

• Pretrained CNN (VGG16) using ImageNet weights.

• Added custom dense layers (Flatten → Dense → Dropout → Softmax).

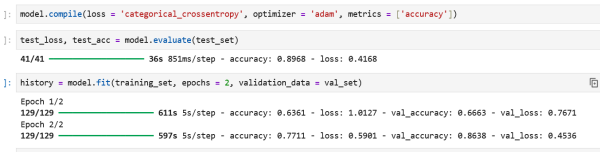
• Achieved strong performance and generalization due to transfer learning.

• Selected as the final model.

Model Development Steps :

Model steps	**Description**
Step 1: Dataset Loading & Preprocessing	Loaded chest X-ray dataset from train, validation, and test folders. Resized all

	images to 64×64, normalized pixel values (rescale 0–1), and generated batches with ImageDataGenerator. [1]: ``` import os os.listdir("chest_xray_covid/Data") ``` [1]: ``` ['test', 'train'] ``` [2]: ``` Image_size = [64, 64] train_path = 'chest_xray_covid/Data/train' test_path = 'chest_xray_covid/Data/test' ```
Step 2: Splitting Dataset	Used ImageDataGenerator with Validation_split = 0.2 to automatically split training data into 80% training and 20% validation. Separate test set used for final evaluation. [3]: ``` import tensorflow as tf from tensorflow import keras from tensorflow.keras.preprocessing.image import ImageDataGenerator # Rescale images train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2 # <-- split 20% of train into validation) test_datagen = ImageDataGenerator(rescale=1./255) # Training set (80% of train data) training_set = train_datagen.flow_from_directory('chest_xray_covid/Data/train', target_size=(64, 64), batch_size=32, class_mode='categorical', subset='training', # <-- important seed=42) ``` ``` # Validation set (20% of train data) val_set = train_datagen.flow_from_directory('chest_xray_covid/Data/train', target_size=(64, 64), batch_size=32, class_mode='categorical', subset='validation', # <-- important seed=42) # Test set (uses separate test folder) test_set = test_datagen.flow_from_directory('chest_xray_covid/Data/test', target_size=(64, 64), batch_size=32, class_mode='categorical') ```
Step 3: Label Encoding	Labels (COVID-19, Normal, Bacteria) were automatically one-hot encoded by flow_from_directory.

	
Step 4: Training the model	Custom CNN trained as baseline. VGG16 Transfer Learning fine-tuned with additional dense layers for classification. 
Step 5: Evaluation	Compared models using training/validation accuracy, loss curves, and confusion matrix. VGG16 achieved higher accuracy and stability, making it the chosen model. 
Step 6: Prediction	The final VGG16-based model was used to predict disease class on unseen test X-ray images.

16]:
```python
from flask import Flask, request, render_template_string
from werkzeug.utils import secure_filename
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
import os

# --------------------------------
# Initialize Flask app
# --------------------------------
app = Flask(__name__)

UPLOAD_FOLDER = "uploads"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# --------------------------------
# Model and class labels
# --------------------------------
class_labels = ["COVID-19", "Normal", "Bacteria"]

try:
    model = load_model("covid.h5")  # replace with your model path
except Exception as e:
    print(f"Error loading model: {e}")
    model = None
```

```python
IMG_SIZE = (64, 64)  # Change to match your model input size

# --------------------------------
# Home route
# --------------------------------
@app.route('/')
def home():
    return render_template_string("""
        <h2>Chest X-ray Classification</h2>
        <form action="/predict" method="post" enctype="multipart/form-data">
            <label>Select an X-ray image:</label>
            <input type="file" name="file" required>
            <button type="submit">Predict</button>
        </form>
    """)

# --------------------------------
# Prediction route
# --------------------------------
@app.route('/predict', methods=['POST'])
def predict():
    if model is None:
        return "Model is not loaded. Please check server logs."

    if 'file' not in request.files:
        return "No file uploaded."
```

```python
    # Save the uploaded file
    filepath = os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(file.filename))
    file.save(filepath)

    try:
        # Preprocess the image
        img = image.load_img(filepath, target_size=IMG_SIZE)
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0) / 255.0  # normalize

        # Predict
        preds = model.predict(img_array)
        predicted_class_idx = np.argmax(preds, axis=1)[0]
        confidence = np.max(preds)
        predicted_label = class_labels[predicted_class_idx]

    except Exception as e:
        return f"Prediction failed: {e}"

    # Return the result
    return render_template_string("""
        <h2>Prediction Result</h2>
        <p><b>Predicted Class:</b> {{ label }}</p>
        <p><b>Confidence:</b> {{ conf | round(2) }}</p>
        <a href="/">Go Back</a>
    """, label=predicted_label, conf=float(confidence))
```

```
# ---------------------------------
# Run app
# ---------------------------------
if __name__ == "__main__":
    # In Jupyter, avoid signal issues
    app.run(debug=False, use_reloader=False)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train
or evaluate the model.
 * 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 http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
```

```
!pip install flask pyngrok
```

```
Requirement already satisfied: flask in c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages (3.1.2)
Requirement already satisfied: pyngrok in c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages (7.2.12)
Requirement already satisfied: blinker>=1.9.0 in c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages (from flask)
(1.9.0)
Requirement already satisfied: click>=8.1.3 in c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages (from flask)
(8.1.8)
Requirement already satisfied: itsdangerous>=2.2.0 in c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages (from flask)
(2.2.0)
Requirement already satisfied: jinja2>=3.1.2 in c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages (from flask)
```

```
import threading

ngrok.kill()


public_url = ngrok.connect(5000)
print(" * Ngrok tunnel URL:", public_url)

def run_app():
    app.run(port=5000)

thread = threading.Thread(target=run_app)
thread.start()
```

```
 * Ngrok tunnel URL: NgrokTunnel: "https://nondilatable-evanescently-jeri.ngrok-free.dev" -> "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 http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug:127.0.0.1 - - [02/Oct/2025 05:31:14] "GET / HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 884ms/step
INFO:werkzeug:127.0.0.1 - - [02/Oct/2025 05:31:27] "POST /predict HTTP/1.1" 200 -
```