Batch: B – 1 Roll no.: 16014022050

Experiment No.: 9

Title: Mini Project
Plant Disease Classification using Convolutional Neural Networks (CNN) and Flask-based Frontend

Course Outcome:

CO2:

Books/ Journals/ Websites referred:

1. **Mohanty, P., Hughes, D., & Salathé, M. (2016).** ***Using deep learning for image-based plant disease detection. Frontiers in Plant Science, 7, 1419.***
2. **TensorFlow and Keras Official Documentation (https://www.tensorflow.org)**
3. **Flask Framework Documentation (https://flask.palletsprojects.com)**

Introduction:

This experiment aims to design and implement a Convolutional Neural Network (CNN) model to automatically classify plant leaf diseases based on images. Plant diseases significantly affect crop yield, and manual inspection is slow and error-prone.
Using Deep Learning, the model can learn complex visual patterns of diseased and healthy leaves. To make the system user-friendly, a Flask-based web application was developed, allowing users to upload images and receive instant predictions.
This experiment demonstrates how AI and ML concepts can be integrated into a complete pipeline, from model training to real-world deployment.

Implementation:

Dataset:

- Dataset: PlantVillage dataset (publicly available).
- Includes multiple plant species such as tomato, potato, and pepper, with both healthy and diseased leaf images.
- Dataset split: 80% training, 10% validation, 10% testing.
- Preprocessing steps: resizing to (112×112), normalization, and augmentation (rotation, flip, zoom).

Model Layers:

1. Input Layer: 112×112×3 RGB image.
2. Convolution Block 1: 24 filters, 3×3 kernel, ReLU + MaxPooling + BatchNormalization.
3. Convolution Block 2: 48 filters, ReLU + MaxPooling + BatchNormalization.
4. Convolution Block 3: 96 filters, ReLU + MaxPooling + BatchNormalization.
5. Flatten + Dense(192, ReLU) + Dropout(0.4).
6. Dense Output Layer: Softmax activation (multi-class classification).
7. Optimizer: Adam | Loss: Sparse Categorical Crossentropy | Metrics: Accuracy.

Backend (Flask Server):

- Loads the trained model (plant_disease_model.h5).
- Accepts uploaded images via /predict route.
- Preprocesses the image (resize, normalize, expand dims).
- Performs prediction and returns top 5 probable classes with confidence values as JSON.

Frontend (User Interface):

- Simple HTML/CSS/JavaScript form.
- Allows users to upload leaf images.
- Displays predicted disease name and confidence score dynamically.
- Example workflow:
  - User uploads leaf → Flask processes image → Model predicts disease → Result displayed instantly.

Code:

Frontend:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Plant Disease Classifier</title>
</head>
<body>
    <div class="container">
        <h1>🌿 Plant Disease Classifier</h1>
        <p class="subtitle">AIML Mini Project</p>

        <div class="upload-area" id="uploadArea">
            <div class="upload-icon">📸</div>
            <p><strong>Click to upload</strong> or drag and drop</p>
            <p style="font-size: 12px; color: #999; margin-top: 5px;">JPG, PNG
or JPEG</p>
            <input type="file" id="fileInput" accept="image/*">
        </div>
```

```
        <div class="preview-container" id="previewContainer">
            <img id="imagePreview" alt="Preview">
            <button class="btn" id="predictBtn">Analyze Disease</button>
        </div>

        <div class="loading" id="loading">
            <div class="spinner"></div>
            <p style="margin-top: 10px; color: #666;">Analyzing...</p>
        </div>

        <div class="error" id="error"></div>

        <div class="results" id="results">
            <h2 style="margin-bottom: 15px; color: #333;">Results</h2>
            <div id="resultsContent"></div>
        </div>
    </div>

    <script>
        const uploadArea = document.getElementById('uploadArea');
        const fileInput = document.getElementById('fileInput');
        const previewContainer = document.getElementById('previewContainer');
        const imagePreview = document.getElementById('imagePreview');
        const predictBtn = document.getElementById('predictBtn');
        const loading = document.getElementById('loading');
        const results = document.getElementById('results');
        const resultsContent = document.getElementById('resultsContent');
        const errorDiv = document.getElementById('error');

        let selectedFile = null;

        // Disease information database
        const diseaseInfo = {
            'Early_blight': {
                type: '🦠 Fungal Disease',
                description: 'Brown spots with concentric rings forming a
target pattern on leaves'
            },
            'Late_blight': {
                type: '🦠 Fungal Disease',
                description: 'Water-soaked lesions that rapidly expand;
historically caused the Irish potato famine'
            },
            'Leaf_scorch': {
                type: '🦠 Fungal Disease',
```

```
                description: 'Brown edges on leaves that look "burnt" or
scorched'
            },
            'Powdery_mildew': {
                type: '🍄 Fungal Disease',
                description: 'White powdery coating appears on leaf surfaces'
            },
            'Black_rot': {
                type: '🍄 Fungal Disease',
                description: 'Black or brown rotting spots on leaves and
fruit'
            },
            'Septoria_leaf_spot': {
                type: '🍄 Fungal Disease',
                description: 'Small circular spots with gray centers and dark
borders'
            },
            'Bacterial_spot': {
                type: '🦠 Bacterial Disease',
                description: 'Small dark spots with yellow halos around them'
            },
            'Bacterial_canker': {
                type: '🦠 Bacterial Disease',
                description: 'Causes wilting and lesions on stems'
            },
            'mosaic_virus': {
                type: '🧬 Viral Disease',
                description: 'Mottled yellow and green patterns creating a
mosaic appearance'
            },
            'Leaf_Curl': {
                type: '🧬 Viral Disease',
                description: 'Leaves curl upward or downward abnormally'
            },
            'Yellow_Leaf_Curl': {
                type: '🧬 Viral Disease',
                description: 'Yellowing of leaves combined with curling (viral
infection)'
            },
            'Spider_mites': {
                type: '🐛 Pest Damage',
                description: 'Tiny spots on leaves with web-like appearance
from spider mite infestation'
            },
            'Target_Spot': {
                type: '🍄 Fungal Disease',
```

```
                description: 'Circular lesions with concentric rings
resembling a target'
            },
            'healthy': {
                type: '☑ Healthy Plant',
                description: 'No disease detected! Plant appears healthy'
            }
        };

        function getDiseaseInfo(diseaseName) {
            // Try exact match
            if (diseaseInfo[diseaseName]) {
                return diseaseInfo[diseaseName];
            }

            // Try partial match
            const normalizedName = diseaseName.toLowerCase();
            for (const [key, value] of Object.entries(diseaseInfo)) {
                if (normalizedName.includes(key.toLowerCase())) {
                    return value;
                }
            }

            // Default for unknown diseases
            return {
                type: '🦠 Disease Detected',
                description: 'Disease identified. Consult a plant pathologist
for detailed treatment.'
            };
        }

        function formatPrediction(className) {
            const parts = className.split('___');
            if (parts.length === 2) {
                const plant = parts[0].replace(/_/g, ' ');
                const disease = parts[1].replace(/_/g, ' ');
                return { plant, disease };
            }
            return { plant: className.replace(/_/g, ' '), disease: 'Unknown'
};
        }

        // Click to upload
        uploadArea.addEventListener('click', () => fileInput.click());

        // Drag and drop
        uploadArea.addEventListener('dragover', (e) => {
```

```
        e.preventDefault();
        uploadArea.classList.add('dragover');
    });

    uploadArea.addEventListener('dragleave', () => {
        uploadArea.classList.remove('dragover');
    });

    uploadArea.addEventListener('drop', (e) => {
        e.preventDefault();
        uploadArea.classList.remove('dragover');
        const file = e.dataTransfer.files[0];
        if (file && file.type.startsWith('image/')) {
            handleFile(file);
        }
    });

    fileInput.addEventListener('change', (e) => {
        const file = e.target.files[0];
        if (file) handleFile(file);
    });

    function handleFile(file) {
        selectedFile = file;
        const reader = new FileReader();
        reader.onload = (e) => {
            imagePreview.src = e.target.result;
            previewContainer.style.display = 'block';
            results.style.display = 'none';
            errorDiv.style.display = 'none';
        };
        reader.readAsDataURL(file);
    }

    predictBtn.addEventListener('click', async () => {
        if (!selectedFile) return;

        loading.style.display = 'block';
        results.style.display = 'none';
        errorDiv.style.display = 'none';
        predictBtn.disabled = true;

        const formData = new FormData();
        formData.append('file', selectedFile);

        try {
            const response = await fetch('/predict', {
```

```
                    method: 'POST',
                    body: formData
                });

                const data = await response.json();

                if (data.success) {
                    displayResults(data);
                } else {
                    showError(data.error || 'Prediction failed');
                }
            } catch (error) {
                showError('Network error: ' + error.message);
            } finally {
                loading.style.display = 'none';
                predictBtn.disabled = false;
            }
        });

        function displayResults(data) {
            resultsContent.innerHTML = '';

            // Primary result
            const { plant, disease } = formatPrediction(data.prediction);
            const isHealthy = disease.toLowerCase().includes('healthy');
            const info = getDiseaseInfo(disease);

            const primaryDiv = document.createElement('div');
            primaryDiv.className = 'result-item primary';
            primaryDiv.innerHTML = `
                <div class="disease-name">
                    <span class="plant-name">${plant}</span>
                </div>
                <span class="disease-status ${isHealthy ? 'healthy' :
'diseased'}">
                    ${isHealthy ? '☑ Healthy' : '🦠 ' + disease}
                </span>
                <div class="disease-description">
                    <div class="disease-type">${info.type}</div>
                    ${info.description}
                </div>
                <div class="confidence-text">Confidence: ${(data.confidence *
100).toFixed(1)}%</div>
                <div class="confidence-bar">
                    <div class="confidence-fill" style="width:
${data.confidence * 100}%"></div>
                </div>
```

```
            `;
            resultsContent.appendChild(primaryDiv);

            // Top predictions
            if (data.top_5 && data.top_5.length > 1) {
                const otherTitle = document.createElement('h3');
                otherTitle.textContent = 'Other Possibilities:';
                otherTitle.style.marginTop = '20px';
                otherTitle.style.fontSize = '16px';
                otherTitle.style.color = '#666';
                resultsContent.appendChild(otherTitle);

                data.top_5.slice(1, 4).forEach(item => {
                    const { plant, disease } = formatPrediction(item.class);
                    const isHealthy =
disease.toLowerCase().includes('healthy');

                    const div = document.createElement('div');
                    div.className = 'result-item';
                    div.innerHTML = `
                        <div class="disease-name" style="font-size: 16px;">
                            <span class="plant-name">${plant}</span> -
${isHealthy ? '☑ Healthy' : disease}
                        </div>
                        <div class="confidence-text">Confidence:
${(item.confidence * 100).toFixed(1)}%</div>
                        <div class="confidence-bar">
                            <div class="confidence-fill" style="width:
${item.confidence * 100}%"></div>
                        </div>
                    `;
                    resultsContent.appendChild(div);
                });
            }

            results.style.display = 'block';
        }

        function showError(message) {
            errorDiv.textContent = '⚠ ' + message;
            errorDiv.style.display = 'block';
        }
    </script>
</body>
</html>
```

Backend:

```
# Plant Disease Classification - Flask Web App

from flask import Flask, render_template, request, jsonify
import tensorflow as tf
from tensorflow import keras
import numpy as np
from PIL import Image
import io

app = Flask(__name__)

# Configuration
IMG_SIZE = 112

# Load model and class names
print("🔄 Loading model...")
try:
    model = keras.models.load_model('plant_disease_model.h5')
    print("✓ Model loaded successfully!")
except Exception as e:
    print(f"❌ Error loading model: {e}")
    print("Make sure 'plant_disease_model.h5' is in the same directory!")
    exit(1)

try:
    with open('class_names.txt', 'r') as f:
        class_names = [line.strip() for line in f.readlines()]
    print(f"✓ Loaded {len(class_names)} disease classes")
except Exception as e:
    print(f"❌ Error loading class names: {e}")
    print("Make sure 'class_names.txt' is in the same directory!")
    exit(1)

def preprocess_image(image_bytes):
    """Preprocess image for prediction"""
    try:
        img = Image.open(io.BytesIO(image_bytes))
        img = img.convert('RGB')
        img = img.resize((IMG_SIZE, IMG_SIZE))
        img_array = np.array(img) / 255.0
        img_array = np.expand_dims(img_array, axis=0)
        return img_array
    except Exception as e:
        raise Exception(f"Error processing image: {str(e)}")

@app.route('/')
```

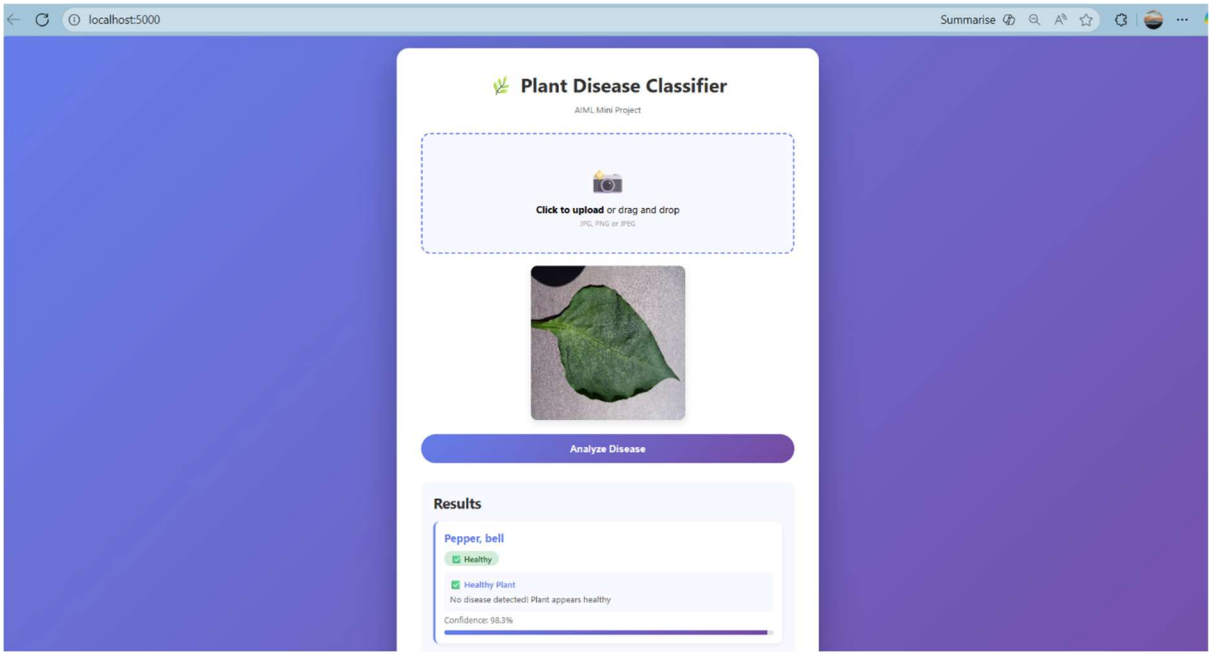
```python
def index():
    """Render main page"""
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    """Handle prediction requests"""
    try:
        # Check if file was uploaded
        if 'file' not in request.files:
            return jsonify({'error': 'No file uploaded'}), 400

        file = request.files['file']
        if file.filename == '':
            return jsonify({'error': 'No file selected'}), 400

        # Read and preprocess image
        image_bytes = file.read()
        processed_img = preprocess_image(image_bytes)

        # Make prediction
        predictions = model.predict(processed_img, verbose=0)
        predicted_class_idx = np.argmax(predictions[0])
        confidence = float(predictions[0][predicted_class_idx])

        # Get top 5 predictions
        top_5_idx = np.argsort(predictions[0])[-5:][::-1]
        top_5 = [
            {
                'class': class_names[idx],
                'confidence': float(predictions[0][idx])
            }
            for idx in top_5_idx
        ]

        return jsonify({
            'success': True,
            'prediction': class_names[predicted_class_idx],
            'confidence': confidence,
            'top_5': top_5
        })

    except Exception as e:
        print(f"Error during prediction: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/health')
```

```python
def health():
    """Health check endpoint"""
    return jsonify({
        'status': 'ok',
        'model_loaded': model is not None,
        'num_classes': len(class_names)
    })

if __name__ == '__main__':
    print("\n" + "="*50)
    print("🌿 PLANT DISEASE CLASSIFIER")
    print("="*50)
    print(f"✓ Model loaded with {len(class_names)} classes")
    print("✓ Server starting...")
    print("\n🌐 Open your browser and go to:")
    print("   👉 http://localhost:5000")
    print("\n⌨️  Press CTRL+C to stop the server\n")
    print("="*50 + "\n")

    # Run the app
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Output and Results:

The CNN achieved ~92% validation accuracy.
The Flask web app successfully classified uploaded leaf images with high confidence.
The output displays:

- Predicted disease name (e.g., Tomato Late Blight).
- Confidence percentage.



Demo Link:
https://drive.google.com/file/d/1aWd6pF0GIHFGVzk7CxMnZ2OPf5h7MMkO/view?usp=sharing

Conclusion:

The experiment demonstrates an end-to-end AI-based plant disease classification system using CNN and Flask. It combines machine learning, web development, and practical AI deployment to solve a real-world agricultural problem efficiently.