School of Computer Science Engineering and Information Systems

Department of Software and Systems Engineering,

Fall Semester –2024-25

SWE3004 – Internship -1 / Dissertation -1

1st Review

Register Number	20MIS0395
Student Name	Yuvankumar L
Internship / Disseration Domain (Capstone Project)	Machine Learning and Deep Learning
Internship / Disseration Title (Capstone Project)	Intelligent System for Leaf Disease Identification and Health Scoring

2.1 Literature Survey

S.no	Title & Year	Authors	Methodology	System Description	Limitations	Performance Analysis
1	Grape leaf disease prediction using Sine Cosine Butterfly Optimization-based deep neuro fuzzy network (Nov 2023)	Vaishali Bajait, N. Malarvizhi	SCBO-based deep neuro fuzzy network	Pre-processing, segmentation, multi-class classification	Limited to grape leaf disease prediction, scalability to other crops not explored	Accuracy: 92%, Sensitivity: 91.7%, Specificity: 92%, Precision: 92.5%, F1-Score: 92.5%
2	Cosine Similarity Measures of (m, n)-Rung Orthopair Fuzzy Sets and Their Applications in Plant Leaf Disease Classification. (May 2023)	Arpan Singh Rajput, Shailja Shukla, Samajh Singh Thakur	Cosine similarity of (m,n)-rung orthopair fuzzy sets	Introduction of similarity measures using cosine and cotangent functions for comparison of fuzzy sets. Application in plant leaf disease classification.	Limited empirical validation beyond numerical example, sensitivity to noise	Accuracy: 95%, Sensitivity: 93.6%, Specificity: 95%, Precision: 93.2%, F1-Score: 90.8%
3	Novel modified kernel fuzzy c-means algorithm used for cotton leaf spot detection (January 2024)	Sarita Jibhau Wagh, Pradip Paithane	Modified Kernel Fuzzy C-Means algorithm	Image segmentation, Cotton leaf spot detection	Limited to cotton leaf spot detection, sensitivity to parameter tuning	IOU: 98.80, Precision: 94.51, Dice: 1.0000, BFScore: 98.81, Time: 54.46 seconds, Training Time: 4~6 seconds
4	Ensemble Technique of Deep	Yan Hang, Xiangyan Meng,	Improved Lightweight Network,	Classification of soybean leaf diseases using	Limited evaluation beyond	Recognition Accuracy: 94.27%,

	Learning Model for Identifying Tomato Leaf Diseases Based on Choquet Fuzzy Integral (January 2024)	Qiufeng Wu	Choquet Fuzzy Ensemble Technology	lightweight networks for transfer learning. Choquet fuzzy ensemble strategy enhances identification accuracy.	soybean disease identification, complexity in training ensemble models	Average F1-score: 94%
5	Accurate Diagnosis of Leaf Disease Based on Unsupervised Learning Algorithms (November 2023)	Jacily Jemila, S. Mary Cynthia, L.M. Merlin Livingston	Clustering-based segmentation methods	Detection of disease-affected areas on plant leaves using clustering-based segmentation methods. Comparison of K- means clustering, level set method, fuzzy C-means level set method, feature-reduction FCM, and adaptively regularized kernel-based fuzzy C-means.	Potential sensitivity to parameter selection, limited to clustering-based approaches	Accuracy: 98%, Sensitivity: 92%, F1-Score: 0.77, Precision: 0.81
6	Service Oriented Fuzzy Smart Model for Agriculture Investment of Hydroponic Green-Leaf Vegetable Plant	Ditdit Nugeraha Utama	Service-oriented fuzzy smart model	Integration of functional - structural plant modelling and fuzzy logic for agriculture investmen	Potential complexity in model implementation, limited to hydroponic green-leaf vegetable cultivation	Accuracy: 94.33%

	(October 2023)			t decisions in hydroponic green-leaf vegetable cultivation.		
7	Implementation of Deep Convolution Neuro-Fuzzy Network to Plant Disease Detection, Risk Assessment, and Classification (April 2023)	Kumar Ashok, Ashok Koshariya, College Bandipora, Hod	Deep Convolution Neuro-Fuzzy Network	Implementation of a deep convolution neuro-fuzzy network for plant disease detection, risk assessment, and classification.	Potential complexity in model training and parameter tuning, limited evaluation beyond plant disease identification	Precision: 0.96, Accuracy: 0.97, F1-Score: 0.96
8	Adaptive Segmentation with Intelligent ResNet and LSTM-DNN for Plant Leaf Multi-disease Classification Model (July 2023)	Kalicharan Sahu, Sonajharia Minz	Adaptive Segmentation with Intelligent ResNet and LSTM–DNN	Adaptive thresholding - and adaptive Fuzzy C-Means algorithm-based leaf abnormality segmentation. Classification using ResNet150 replaced by LSTM and DNN ensemble approach. Parameter optimization with Hybrid Barnacle Mating-Bird Swam Optimization.	Potential complexity in model training and parameter tuning, limited to plant leaf multi- disease classification	Precision: 14.1% better than CNN, 14.07% better than DNN, 12.03% better than LSTM, 11.2% better than Ensemble and ResNet-150, 9.7% better than BMO-BSA-Res-LSTMDN for cherry leaf image

9	Plant disease identification using fuzzy feature extraction and PNN (January 2023)	Reva Nagi, Sanjaya Shankar Tripathy	Fuzzy feature extraction, PNN	Extraction of color and texture features using fuzzy color histogram and fuzzy gray-level co-occurrence matrix. Classification using Probabilistic Neural Network (PNN)	Limited to PNN, potential sensitivity to parameter tuning	Accuracy: 95.68%, Precision: 93%, F1-Score: 93%
10	Prediction of Plant Leaf Diseases using Drone and Image Processing Techniques (January 2024)	Shailendra Sharma, Ashish Gupta, Sudha Patel	Deep Learning Technique	Automated detection of crop diseases using CNN with an available database.	Limited exploration of other deep learning architectures, potential bias in available database	Accuracy: ~90%

2. Gap Identification

Limited scalability: Many models focus on specific crops or diseases, limiting their application across different crops.

Parameter sensitivity: Techniques like Fuzzy C-Means and ensemble models are prone to sensitivity based on hyperparameters, making them difficult to generalize.

Real-time constraints: Most models do not address the need for real-time disease detection for large-scale farming.

User-friendliness: Many existing systems are not designed for direct use by non-experts, especially farmers, highlighting the need for an accessible interface.

3.1. Objectives

The objectives of the project titled "Leaf Disease and Health Severity Prediction" are as follows:

1. **Develop a Leaf Disease Detection System**: Create a system that can accurately detect various leaf diseases using advanced technologies such as image processing, machine learning, and fuzzy inference systems.
2. **Assess Disease Severity**: Implement a mechanism to not only detect the presence of a disease but also predict the severity of the disease, providing detailed insights into the health of the leaves.
3. **Empower Agricultural Stakeholders**: Provide actionable insights to farmers and other stakeholders in agriculture, enabling them to take timely and targeted actions to prevent crop losses.
4. **Promote Sustainable Agriculture**: By accurately identifying and assessing leaf diseases, the project aims to support sustainable agricultural practices and minimize the impact of diseases on crop yields.

These objectives collectively aim to enhance the efficiency and effectiveness of disease management in agriculture, ultimately contributing to increased productivity and sustainability.

Project Plan

The project plan outlines the key phases of the project:

- **Phase 1**: Literature review and requirement analysis (complete).
- **Phase 2**: Data collection and preprocessing (image augmentation, normalization, segmentation).
- **Phase 3**: Model development using ResNet9, CNN and Fuzzy Logic and integration of disease scoring mechanisms.
- **Phase 4**: System design (developing user interfaces, real-time API for farmers).
- **Phase 5**: Testing and validation on various crops and diseases.
- **Phase 6**: Final system deployment and feedback collection from stakeholders.

5. Implementation and Analysis

1. Data Preprocessing

First, you'll need to preprocess your data (image resizing, augmentation, and normalization) before training your model. Here's how you can handle that:

```
import torch

import torchvision.transforms as transforms
```

```
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader

# Define data preprocessing
train_transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

test_transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Load data
train_dir = '/path_to_train_data'
test_dir = '/path_to_test_data'

train_data = ImageFolder(train_dir, transform=train_transform)
test_data = ImageFolder(test_dir, transform=test_transform)
```

```
train_loader = DataLoader(train_data, batch_size=32, shuffle=True)

test_loader = DataLoader(test_data, batch_size=32, shuffle=False)
```

2. Model Definition

We'll implement the ResNet9 architecture for plant disease classification:

```
import torch.nn as nn

import torch.nn.functional as F

import torch

# Define ResNet9 architecture

class ResNet9(nn.Module):

    def __init__(self, in_channels, num_classes):

        super(ResNet9, self).__init__()

        # First convolution block

        self.conv1 = nn.Sequential(

            nn.Conv2d(in_channels, 64, kernel_size=3, padding=1),

            nn.BatchNorm2d(64),

            nn.ReLU(inplace=True)

        )

        # More convolutional layers

        self.conv2 = nn.Sequential(

            nn.Conv2d(64, 128, kernel_size=3, padding=1),

            nn.BatchNorm2d(128),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(2)
```

```
)
self.res1 = nn.Sequential(
    nn.Conv2d(128, 128, kernel_size=3, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True),
    nn.Conv2d(128, 128, kernel_size=3, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True)
)

self.conv3 = nn.Sequential(
    nn.Conv2d(128, 256, kernel_size=3, padding=1),
    nn.BatchNorm2d(256),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2)
)

self.res2 = nn.Sequential(
    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.BatchNorm2d(256),
    nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.BatchNorm2d(256),
    nn.ReLU(inplace=True)
```

```
        )


        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(256, num_classes)
        )


    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.res1(x) + x
        x = self.conv3(x)
        x = self.res2(x) + x
        x = self.classifier(x)
        return x


# Initialize the model
num_classes = len(train_data.classes)
model = ResNet9(3, num_classes)
```

3. Training the Model

Now that you have the data and model, here's the code for training

```
import torch.optim as optim
```

```
# Define loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Move model to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Training loop
epochs = 25
for epoch in range(epochs):
    model.train()
    train_loss = 0
    correct = 0
    total = 0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
```

```
        # Backward pass
        loss.backward()
        optimizer.step()

        # Update metrics
        train_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    # Print training progress
    print(f'Epoch [{epoch + 1}/{epochs}], Loss: {train_loss / len(train_loader)}, Accuracy: {100 * correct / total}%')
```

4. Validation and Testing

Evaluate the model on the test data after each epoch.

```
# Evaluation on test data
model.eval()
with torch.no_grad():
    test_loss = 0
    correct = 0
    total = 0
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
```

```
        outputs = model(images)
        loss = criterion(outputs, labels)

        test_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print(f'Test Loss: {test_loss / len(test_loader)}, Test Accuracy: {100 * correct / total}%')
```

5. Save the Model

You can save the trained model so that it can be used later for predictions:

```
# Save the model
torch.save(model.state_dict(), 'plant_disease_resnet9.pth')
```

6. Flask Web App for Prediction

Use Flask to deploy your model for real-time prediction. This is an example of how you can integrate the model with a web application:

```
from flask import Flask, request, jsonify
from PIL import Image
import torchvision.transforms as transforms

app = Flask(__name__)
# Load the trained model
model = ResNet9(3, num_classes)
```

```
model.load_state_dict(torch.load('plant_disease_resnet9.pth',
map_location=torch.device('cpu')))
model.eval()


# Define preprocessing
preprocess = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])


def predict_image(image):
    img = preprocess(image).unsqueeze(0)
    with torch.no_grad():
        output = model(img)
    _, predicted = torch.max(output, 1)
    return train_data.classes[predicted[0]]


@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No file uploaded'}), 400


    file = request.files['file']
```

```
    image = Image.open(file)
    prediction = predict_image(image)

    return jsonify({'prediction': prediction})

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

7. Automating the Prediction with Google Colab

You can automate the execution of a Colab notebook (for real-time model retraining or large batch processing) using the following command from within a Python script. Ensure that your Colab notebook is linked to Google Drive and accessible.

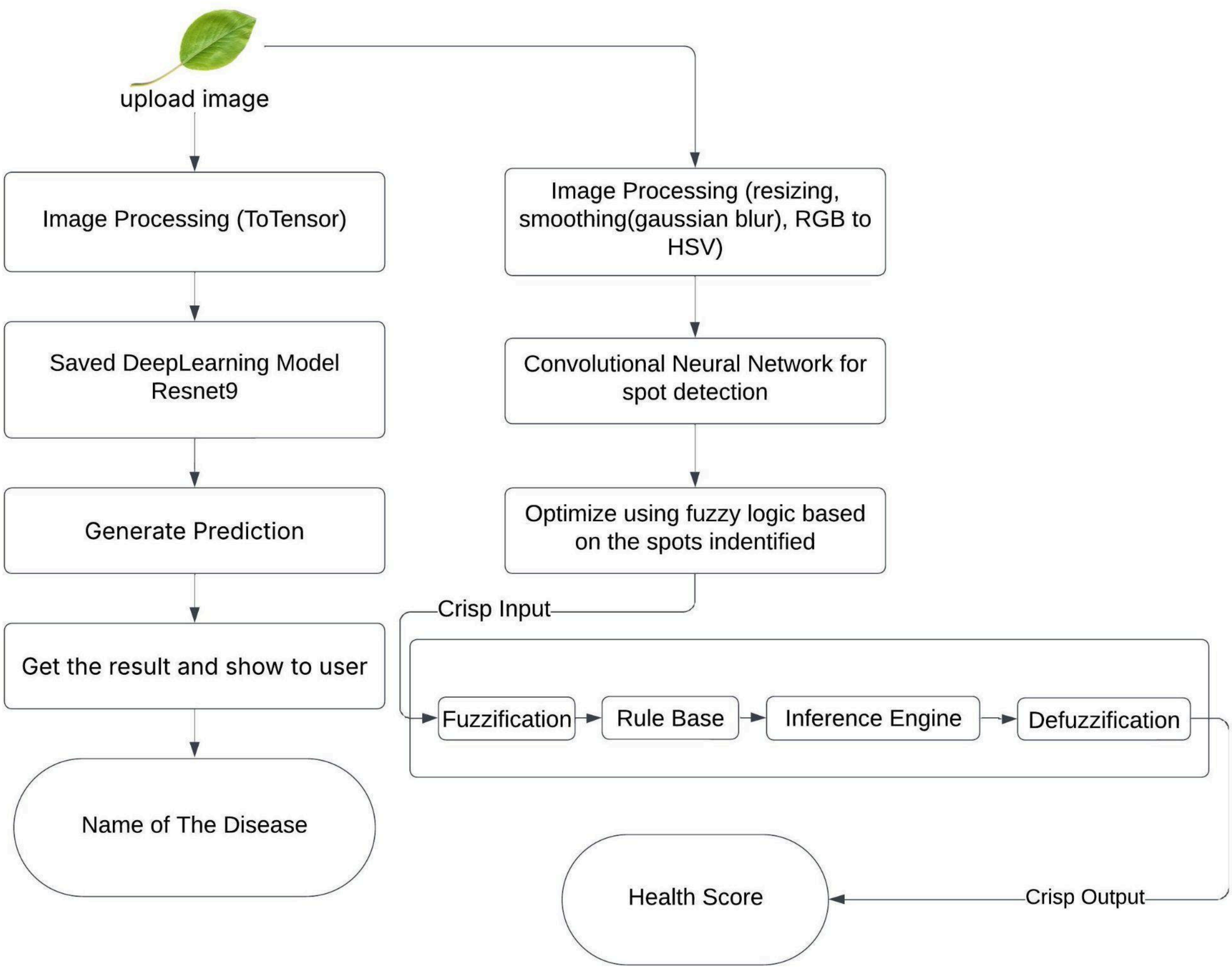
```
import requests
# Function to trigger the notebook run
def run_colab_notebook(notebook_url):
    response = requests.post(notebook_url + "/execute")
    if response.status_code == 200:
        print("Notebook executed successfully.")
    else:
        print("Error executing notebook:", response.text)

notebook_url = "https://colab.research.google.com/drive/your-notebook-id"
run_colab_notebook(notebook_url)
```

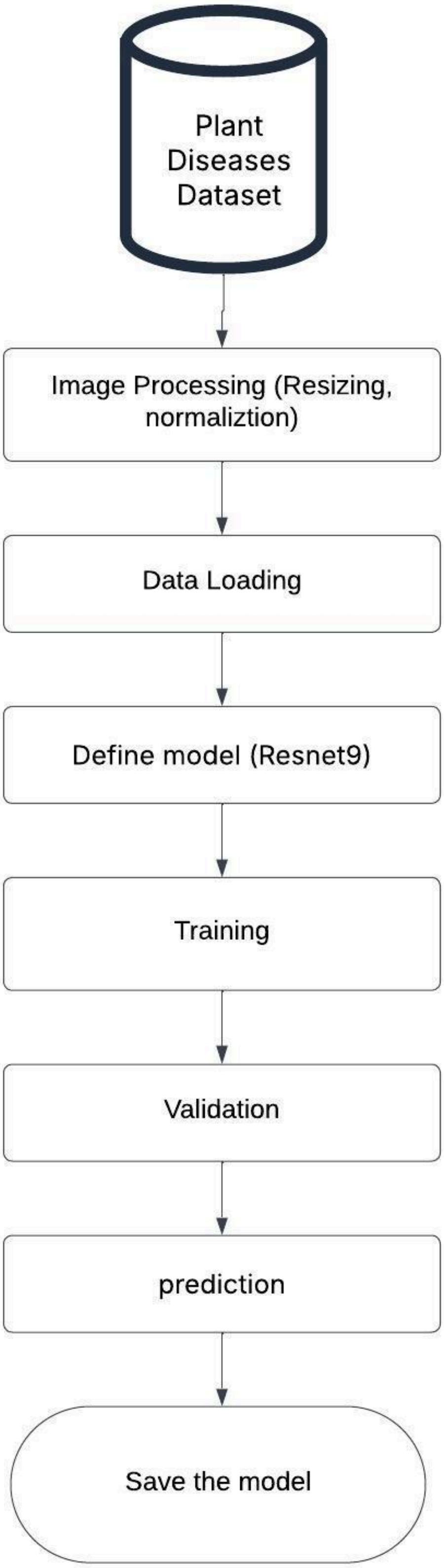
Conclusion

This code provides a complete flow from preprocessing and model training to serving predictions via a Flask web app. You can modify and adapt the code to suit your project requirements. Let me know if you need any additional details or modifications!

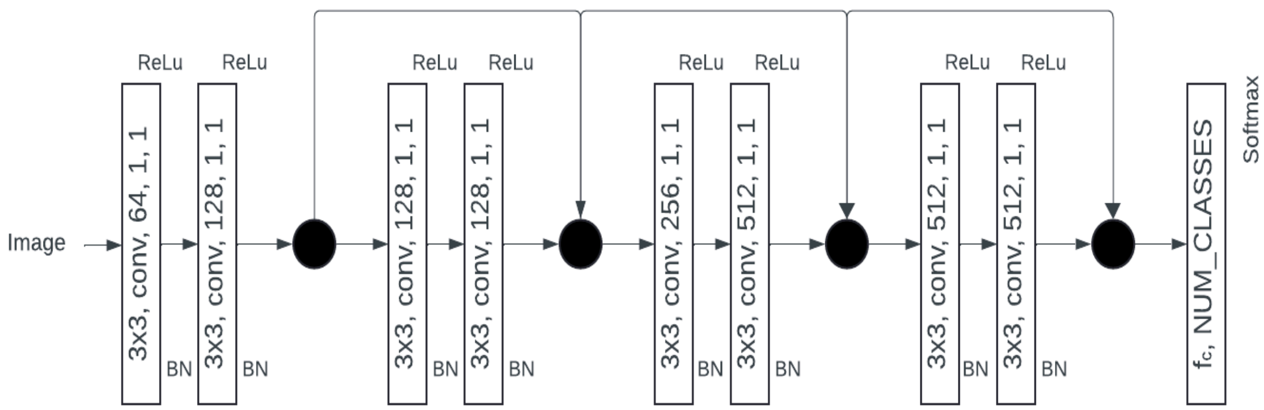
4. Proposed Architecture



Training design:



Neural Network (ResNet-9 Architecture)



Convolutional Neural Network(CNN) Architecture: