# Mango Leaf Disease Detection – Two‑Stage Model

This repository contains the Python reference implementation that accompanies the paper *“A Two‑Stage Model for Enhanced Mango Leaf Disease Detection Using an Innovative Handcrafted Spatial Feature Extraction Method and Knowledge Distillation Process”*. The objective of the project is to detect diseases on mango leaves using a lightweight yet accurate pipeline. A two‑stage strategy is adopted:

* Handcrafted feature extraction and baseline classifiers. Each leaf image is resized to a common resolution and converted to grayscale, after which Local Directional Patterns (LDP) and their enhanced variant Local Directional Pattern variance (LDPv) are computed. These methods capture directional edge and contrast information, and the image is divided into fixed‑size blocks; mean values within each block are used to summarise local texture patterns. The resulting feature vectors are used to train traditional classifiers. Support Vector Machine (SVM) and K‑Nearest Neighbour (KNN) classifiers are chosen because they perform well on texture features.
* Knowledge distillation (KD). A second stage trains a large “teacher” neural network on the handcrafted feature vectors and then transfers its knowledge to a smaller “student” network. The KD process uses soft targets from the teacher to guide the student, minimises Kullback–Leibler divergence between their output distributions and incorporates a class‑wise penalty to reduce false negatives. This allows the student model to approach teacher‑level performance while retaining a small footprint suitable for deployment on resource‑constrained devices. In experiments the KD model consistently outperformed the baseline SVM and KNN models; for example, in the “Combined” configuration for Bacterial Canker at a 9×9 block size, the KD model achieved 96.95 % accuracy with a false‑negative rate of only 0.27 %.

The handcrafted feature extraction scripts are provided in MATLAB (see README\_MatLab.md), while this README focuses on the Python code used to train and evaluate the baseline and KD models.

## Dataset

The experiments rely on the MangoLeafBD dataset, a publicly available collection of 4 000 colour images of mango leaves captured in Bangladeshi orchards. Each image has a resolution of 240 × 320 pixels, and the dataset contains eight balanced classes (seven disease categories plus a healthy class); each class includes 500 images[[1]](https://data.mendeley.com/datasets/hxsnvwty3r/1#:~:text=Type%20of%20data%3A%20240x320%20mango,Captured%20from%20mango%20trees%20through). The disease classes are *Anthracnose*, *Bacterial Canker*, *Cutting Weevil*, *Die Back*, *Gall Midge*, *Powdery Mildew* and *Sooty Mould*[[1]](https://data.mendeley.com/datasets/hxsnvwty3r/1#:~:text=Type%20of%20data%3A%20240x320%20mango,Captured%20from%20mango%20trees%20through). Images were captured with mobile phone cameras in multiple orchards and augmented by zooming and rotation to increase variety[[1]](https://data.mendeley.com/datasets/hxsnvwty3r/1#:~:text=Type%20of%20data%3A%20240x320%20mango,Captured%20from%20mango%20trees%20through). The dataset may be downloaded from Mendeley Data:

https://data.mendeley.com/datasets/hxsnvwty3r/1

After downloading, organise the images into subdirectories by class. The MATLAB scripts provided in this repository extract LDP/LDPv features from these images and save them in a .mat file. To use the Python code, export the feature matrix and labels to a CSV or NumPy file (see below).

## Requirements

The Python code was developed for Python 3.8 or later and relies on the following libraries:

* NumPy and Pandas for numerical operations and data handling.
* Scikit‑learn for machine‑learning algorithms, hyper‑parameter tuning and evaluation metrics.
* TensorFlow 2.x (with Keras) for building neural networks and implementing knowledge distillation.

Install the dependencies with pip:

pip install numpy pandas scikit-learn tensorflow==2.\*

On systems with limited resources you may wish to install the CPU‑only build of TensorFlow (tensorflow-cpu).

## 

## Repository structure

The key Python scripts and their roles are summarised below.

| File | Role |
| --- | --- |
| Baseline\_Model\_Compare.py | Demonstration of baseline SVM/KNN evaluation with dummy data. |
| SVM.py | Hyper‑parameter tuning and training of an SVM classifier using GridSearchCV. |
| KNN.py | Hyper‑parameter tuning and training of a KNN classifier. |
| KD\_3.1.py | Defines and trains a teacher multi‑layer perceptron (MLP) model. |
| KD\_3.3.py | Defines the student model and implements custom knowledge‑distillation loss with class‑wise false‑negative penalties, then trains the student using soft labels from the teacher. |
| Eval\_KD\_3.4.py | Evaluates the trained student model on a test set and computes accuracy and false‑negative rate. |
| Final\_4.1.py | Contains reusable functions for computing accuracy and false‑negative rate. |
| Final\_4.2.py | Generates overall and class‑wise metrics (accuracy, sensitivity, specificity and FNR) for SVM, KNN and KD models. |
| Final\_4.3.py | Produces detailed class‑wise metrics tables for reporting. |

In addition, README\_MatLab.md documents the MATLAB feature‑extraction pipeline, and .m files implement LDP/LDPv calculations.

## Preparing the feature matrix

1. Extract features using the MATLAB scripts (img\_resize.mat, rgb\_gray.mat, masking\_pic.mat, LDP.mat, LDPv.mat, datasetCode.mat, and concate\_allfeature.mat). These scripts resize images, convert them to grayscale, compute the LDP and LDPv edge responses, divide each response image into blocks (3×3, 6×6 and 9×9) and compute the mean of each block to form a feature vector. The resulting .mat file will contain a matrix of shape (n\_samples, n\_features) and a label vector of length n\_samples.
2. Export to CSV or NumPy. In MATLAB you can use writematrix to export the features and labels:

% Load the .mat file produced by concate\_allfeature.mat  
load('all\_features.mat'); % variable names depend on your script  
writematrix(all\_features, 'features.csv');  
writematrix(labels, 'labels.csv');

Alternatively, you can write a Python script using scipy.io.loadmat to load the .mat file and save the arrays with numpy.save.

1. Load in Python. Within the Python scripts replace the dummy data generation with code to load the exported features:

import pandas as pd  
X = pd.read\_csv('features.csv').values  
y = pd.read\_csv('labels.csv').values.ravel()  
# Optionally scale features  
from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
X = scaler.fit\_transform(X)

## Running the baseline models (Stage 1)

1. SVM (Script SVM.py) — This script performs a grid search over SVM hyper‑parameters (C and gamma for the radial basis function kernel) using five‑fold cross‑validation, as suggested in the paper for high‑dimensional feature sets. It trains the best estimator on the training data and reports test accuracy and false‑negative rate. To use your own data, replace the dummy data block with the X and y arrays obtained from the feature‑extraction step.
2. KNN (Script KNN.py) — This script tunes the number of neighbours and the distance metric via GridSearchCV and trains the best KNN model on your data. As with the SVM script, replace the dummy data with your feature matrix and labels.
3. Comparing baselines — Optionally run Baseline\_Model\_Compare.py, which illustrates how to evaluate the final SVM and KNN models on a test set. For a rigorous comparison, compute overall accuracy and false‑negative rate using the functions provided in Final\_4.1.py or by calling accuracy\_score and confusion\_matrix directly from Scikit‑learn.

## Training the knowledge‑distilled student (Stage 2)

Knowledge distillation improves the model by transferring knowledge from a complex teacher network to a smaller student network. The scripts assume that you already have the feature matrix (X\_train, X\_test) and labels (y\_train, y\_test) prepared.

1. Teacher model (KD\_3.1.py) — Defines a multilayer perceptron (MLP) with several dense layers and dropout for regularisation. It uses categorical cross‑entropy loss and trains on the one‑hot encoded labels. Adjust the network depth, number of units and number of epochs based on your resource budget and target accuracy. After training, the script obtains “soft” probability predictions on the training set. These predictions encode inter‑class relationships that the student will learn.
2. Student model and KD loss (KD\_3.3.py) — Defines a smaller MLP student network and a custom KDModel class that implements the training loop with three loss components:
3. Hard loss: categorical cross‑entropy between the student’s predictions and the true labels.
4. Soft loss: Kullback–Leibler divergence between the temperature‑scaled outputs of the teacher and student networks, weighted by 1 – alpha.
5. False‑negative penalty: a class‑wise penalty term that discourages the student from assigning low probability to the correct class. The penalty weights (lambda\_i) and threshold (delta) correspond to Equation 12 of the paper and can be adjusted to balance sensitivity across classes.

The script trains the student using the teacher network as a fixed component and allows the user to set hyper‑parameters such as the distillation temperature and the alpha weighting between hard and soft losses. Higher temperatures produce softer probability distributions and encourage the student to learn fine‑grained relationships.

1. Evaluation (Eval\_KD\_3.4.py) — After training, this script evaluates the KD‑enhanced student on a hold‑out test set. It computes accuracy and the false‑negative rate (treating the healthy class as negative and all disease classes as positive). The evaluation can be customised by modifying the calculate\_fnr function.

## Measuring performance

Final\_4.1.py defines two utility functions, calculate\_accuracy and calculate\_false\_negative\_rate, which compute overall accuracy and false‑negative rate from arrays of true and predicted labels. These metrics correspond to the definitions used in the paper. The false‑negative rate is particularly important for disease detection, as it quantifies the proportion of diseased leaves incorrectly classified as healthy.

To reproduce the tables in the paper, run Final\_4.2.py to compute overall and class‑wise metrics (accuracy, sensitivity, specificity and FNR) for each model. The script expects the true labels (y\_test) and the predicted labels from your trained models (y\_pred\_svm, y\_pred\_knn and y\_pred\_kd\_student). It outputs human‑readable tables similar to Tables 2 and 3 in the paper, demonstrating that the KD model attains higher accuracy and lower false‑negative rates than the baseline SVM and KNN classifiers. Final\_4.3.py provides an alternative format for reporting per‑class metrics.

## Reproducibility tips

* Random seeds: Use random\_state in train\_test\_split and set seeds for NumPy and TensorFlow to obtain reproducible splits and model initialisations.
* Feature scaling: Standardising features improves neural‑network convergence. Always fit the scaler on the training data and apply it to both training and test sets.
* Hyper‑parameters: When tuning SVM and KNN models, start with parameter grids similar to those used in the scripts. For SVM, the paper recommends the radial basis function kernel because it handles high‑dimensional features effectively. For KNN, exploring odd values of k and different distance metrics (Euclidean, Manhattan) is advisable.
* Class weights and FNR penalties: If certain diseases are more critical to detect, assign higher penalty weights in KD\_3.3.py to reduce false negatives for those classes. Adjust the threshold delta to control when penalties are applied.

## Citation

If you use this code or the accompanying methodology in your research, please cite the original paper:

Islam MM, et al. “A Two‑Stage Model for Enhanced Mango Leaf Disease Detection Using an Innovative Handcrafted Spatial Feature Extraction Method and Knowledge Distillation Process.” *Ecological Informatics* (2025).

You should also cite the MangoLeafBD dataset if it contributes to your work.

## License

The Python code in this repository is released under the MIT Licence. The MangoLeafBD dataset is made available under the CC BY‑NC 3.0 licence[[1]](https://data.mendeley.com/datasets/hxsnvwty3r/1#:~:text=Type%20of%20data%3A%20240x320%20mango,Captured%20from%20mango%20trees%20through). Please review the dataset licence before using it in commercial applications.

[[1]](https://data.mendeley.com/datasets/hxsnvwty3r/1" \l ":~:text=Type%20of%20data%3A%20240x320%20mango,Captured%20from%20mango%20trees%20through) MangoLeafBD Dataset - Mendeley Data

<https://data.mendeley.com/datasets/hxsnvwty3r/1>