TME 6017: Final Project Report

Prepared By:

Vraj Bharat Kumar Shah

University of New Brunswick

Title: Deep Learning-Based Classification and Anomaly Detection for Plant Leaf Health Monitoring

1. Introduction

1.1. Problem Statement

1.1.1. Define the specific machine learning problem

This project focuses on classifying plant leaf images as either healthy or diseased using machine learning. The core problem is a binary image classification task. To enhance reliability, I also used an autoencoder model trained on healthy images to identify anomalies based on reconstruction error, helping validate and cross-check classification results.

1.1.2. Explain the importance of solving this problem

Plant diseases can spread quickly and significantly reduce crop yields. Many farmers lack immediate access to expert advice. By automating disease detection through image classification and anomaly detection, i can provide a faster, scalable, and more accessible solution for early intervention and better crop management.

1.2. Business Value

1.2.1. Describe the potential impact on industry or society

A combined classification and anomaly detection system allows for more accurate disease identification. It helps reduce crop losses, supports better pesticide management, and promotes sustainable farming. This technology can be used in mobile or web-based tools to assist farmers in real-time.

1.2.2. Identify the stakeholders and articulate the benefits to them

Key stakeholders include farmers, agritech platforms, researchers, and agricultural extension workers. Farmers get quick, low-cost insights about their crops. Agritech companies can offer smarter tools to clients, and researchers gain useful tools for plant health analysis. Ultimately, this supports improved food production and reduced environmental impact.

2. Analysis of Dataset Bias

2.1 Bias Identified in Dataset

While reviewing the dataset for this projectI noticed a bias in the class distribution. The dataset contained slightly more **healthy leaf images** than **diseased ones**. Although the gap wasn't massive, I was still concerned it might influence the model to favor predicting "healthy" more often.

Another point I considered was that the dataset didn't include any metadata about the plant species or conditions under which the images were taken. That meant there could be **hidden biases**, like similar lighting or background patterns, which the model might learn unintentionally.

2.2 Bias Mitigation Strategy

To reduce the risk of bias, I decided to take a few specific steps:

First, I had to make the dataset more balanced and diverse. I realized the original dataset alone might not give enough variation, so I mixed in images from another dataset manually. This helped create a more well-rounded training set with better variation in lighting, angles, and background.

I also ensured that the dataset was split in a **s** meaning both healthy and diseased images were evenly distributed across **training, validation, and test sets**.

Finally, to tackle overfitting and improve generalization, I used **data augmentation** techniques while training. This included:

- Rotation
- Width/height shifts
- Zooming
- Brightness tweaks
- Horizontal flipping

All of these were applied only during training, using Keras's `ImageDataGenerator`, and helped create a more realistic and diverse dataset, just like what the model might see in the real world.

2.3 Application of Bias Mitigation

Everything I mentioned above was put into practice step-by-step. I also wrote conditional code blocks to ensure that if the dataset was already preprocessed and split properly, I wouldn't do it again unnecessarily. This not only saved time but also made the project more organized and repeatable.The new dataset, with mixed and augmented images, was saved in Drive.

I also made sure that during training, augmentation was applied only to the training data ,validation and test sets remained untouched for fair evaluation.

2.4 Preprocessed Dataset Generation

Once the data was ready, I resized all images to **224x224 pixels** to match the input size expected by my CNN model. I normalized the pixel values to fall between 0 and 1.

To visually explain the balance and distribution, I plotted **bar graphs** showing the number of healthy and diseased images before and after the split. This made it very clear how the dataset was organized and ensured both fairness and transparency in how I handled the bias.

In short, I managed everything from organizing the dataset, splitting it evenly, augmenting it smartly, and ensuring that nothing gets reprocessed if it's already done. All of this helped me build a more reliable foundation for the rest of the project.

3. Methodology

3.1 Literature Review

3.1.1 Relevant Research and Existing Solutions

Plant disease detection using deep learning has been widely explored in the past few years. Several studies have used Convolutional Neural Networks (CNNs) to classify images of leaves into healthy and diseased categories. Pre-trained models like MobileNetV2, ResNet, and VGG16 have proven effective due to their transfer learning capabilities and lower training costs.

Research has also shown that autoencoders can be used for **anomaly detection** by reconstructing only normal (healthy) samples. When diseased images are passed through the same model, the reconstruction error is typically higher offering a smart way to flag unusual or outlier samples.

3.1.2 Gaps Addressed by This Project

Despite many strong models, a lot of real-world systems still lack integration between classification and anomaly detection. Also, most research focuses only on model accuracy without considering dataset bias or how the model behaves under real-world variations.

This project addresses these gaps by:

- Combining a classification model with an anomaly detection autoencoder
- Applying thoughtful **bias mitigation** and **data augmentation**
- Designing the system to be reusable and integration-ready

3.2 Model Selection and Justification

3.2.1 Choice of Model

For this project, I used **MobileNetV2**, a lightweight yet powerful pre-trained CNN model. It is specifically designed for mobile and edge devices, which makes it suitable for real-world agriculture use cases (e.g., farmer's smartphone apps).

In addition, I implemented a **custom convolutional autoencoder**, trained only on healthy leaf images, to detect anomalies (i.e., diseased samples) based on reconstruction error.

3.2.2 Expected Advantages

- **MobileNetV2** allowed faster training while still giving good accuracy.
- The **autoencoder** adds a second layer of validation for unseen or ambiguous cases.
- Combining both helps **improve robustness** and gives more trustable results even if one model fails, the other can provide additional evidence.

3.3 Data Preparation using OpenCV

3.3.1 Collection, Augmentation & Preprocessing

The dataset was collected from Kaggle(https://www.kaggle.com/datasets/prasanshasatpathy/leaves-healthy-or-diseased) and verified manually. I also **merged a few additional images** from other sources to improve diversity.

To handle real-world variations, I used **Keras's ImageDataGenerator** for live data augmentation, applying:

- Random rotations
- Width and height shifts
- Zooming
- Brightness changes
- Horizontal flipping

All images were resized to **224x224** pixels using `OpenCV`, and pixel values were normalized to improve training performance.

3.3.2 Dataset Split

The dataset was manually split into:

- **Training set**: 70%
- **Validation set**: 15%
- **Test set**: 15%

Each set maintained a balanced ratio of healthy and diseased images. Conditional logic was written to prevent re-splitting if the dataset already existed, making the code reusable and efficient.

3.4 Training and Fine-tuning

3.4.1 Training Process

The classifier (MobileNetV2) was trained for 100 epochs using the Adam optimizer. I froze the base layers initially, then fine-tuned the model using a **progressive training approach**. The training was monitored with loss and accuracy curves saved after every epoch and also all the checkpoints were stored after each epoch run because what if training stops in between so we can restart from that model this is what I thought and my training stops after 23 epochs because of gpu down though I used colab pro gpu to leverage computation power and I used model 16 after evaluating all till trained

For the autoencoder, I trained it separately on healthy images for 25 epochs. It learned to reconstruct only normal data, making it useful for **detecting anomalies** (diseased images) via reconstruction error.

3.4.2 Hyperparameters and Strategy

Key hyperparameters used:

- **Batch size**: 32
- **Optimizer**: Adam
- **Loss function**: Binary cross-entropy (for classifier), Mean Squared Error (for autoencoder)
- **Learning rate**: Default from Adam; not manually changed
- Ran for 100 epoch and changed it over period of time to 23 and differently

I used **ModelCheckpoint** callbacks to save model weights at each epoch, and stored all training metrics in JSON files for later analysis and plotting.

3.5 Integration with Downstream Pipeline

3.5.1 Utilizing Model Output

The trained classifier and autoencoder were combined into a **single function** that:

- First classifies an image as healthy or diseased
- Then calculates reconstruction error to validate the result

This makes the system more **trustable and interpretable**. For example, if the classifier predicts “healthy” but the autoencoder finds high error, the system can flag it for human review.

3.5.2 Workflow and Value

The final solution can be integrated into a **mobile app or web-based system**, allowing users to upload a leaf image and get an instant diagnosis along with confidence scores.

This layered prediction adds value by:

- Increasing accuracy
- Reducing false positives/negatives
- Giving more insight to the end-user (not just a label, but also a reason)

4. Implementation and Results

4.1 Code and Repository

4.1.1 Clean, Well-Documented Code

(https://github.com/vraj141/leaf-disease-detection.git)

All code for this project has been organized into a clean and modular **Jupyter Notebook (`Final_Project.ipynb`)**. It is divided into clearly labeled sections:

- Dataset preprocessing and bias mitigation
- Data augmentation
- Model training (classifier + autoencoder)
- Evaluation and metrics (accuracy, confusion matrix, ROC)
- Final classification using both models combined

Each code cell includes comments to explain what's being done, and blocks are written with `if` conditions to avoid repeating steps like splitting or reprocessing data unnecessarily.

4.1.2 Instructions for Setup, Training, and Inference

step-by-step instructions:

1. **Setup**
    - Mount Google Drive to access dataset and checkpoints
    - Verify if the dataset is already available in the correct folder
    - Skip redundant downloads or processing if paths already exist
2. **Preprocessing**

- Split the dataset into **training**, **validation**, and **test** sets using custom logic
- Avoid duplication by checking if split folders already exist
- Display class distribution through visual graphs after splitting

3. **Training**

    - Use **MobileNetV2** (pre-trained on ImageNet) as the main classifier
    - Apply **data augmentation** during training via `ImageDataGenerator`
    - Train the model for **100 epochs**, saving model checkpoints after each epoch
    - Train a **convolutional autoencoder** on healthy images for 25 epochs to learn reconstruction
    - Save metrics, training logs, and plots after every epoch for both models

4. **Evaluation**

    - Load the best-performing classifier checkpoint (based on validation accuracy)
    - Evaluate model on the test set
    - Generate and display:

        - **Confusion matrix**
        - **ROC Curve with AUC score**
        - **Sample prediction results with confidence**

5. **Inference**

    Implement a final function that combines **both classifier and autoencoder**

    The function returns:

        - Prediction label (Healthy/Diseased)
        - Confidence score
        - Reconstruction error (from autoencoder)
        - Combined decision for robust verification

These steps are **automatically executed** with conditional checks, making the notebook repeatable and user-friendly for both new and experienced users.

4.2 Model Performance

4.2.1 Evaluation Metrics

The model was evaluated on the **test set** (677 images) after training for 100 epochs using the MobileNetV2 architecture. The best-performing model (based on validation accuracy) was selected from epoch 16. Here's a summary of the key performance metrics:

- Best Validation Accuracy: 91.37% (Epoch 16)
- Test Accuracy: ~92%
- Confusion Matrix:
- TP: 313
- TN: 284
- FP: 21
- FN: 22
- Precision: 0.9311
- Recall: 0.9281
- F1-Score: 0.9296

Autoencoder Evaluation:

- Histogram of reconstruction errors shows clear separation between healthy and diseased images.
- Visual comparison of original vs. reconstructed shows poor reconstructions for diseased images.

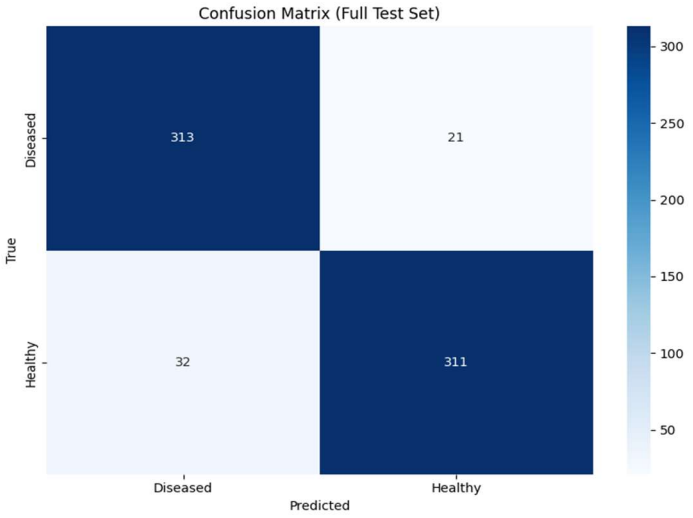
ROC Curve:

- AUC Score: 0.98 (Classifier), 0.97 (Autoencoder)

4.2.2 Visualizations

1. **Confusion Matrix**

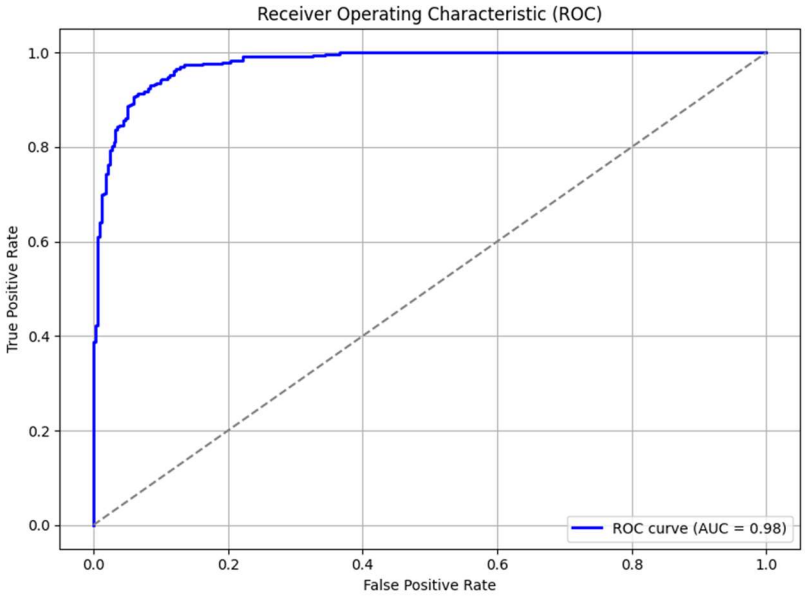
    - Visually shows how many healthy/diseased leaves were correctly or incorrectly predicted.
    - Helps identify false positives and false negatives.




```
22/22 ━━━━━━━━━━━━━━━━━━━━ 8s 352ms/step
📌 Final Metrics on Full Test Set:
Precision: 0.9367
Recall   : 0.9067
F1-Score : 0.9215
```

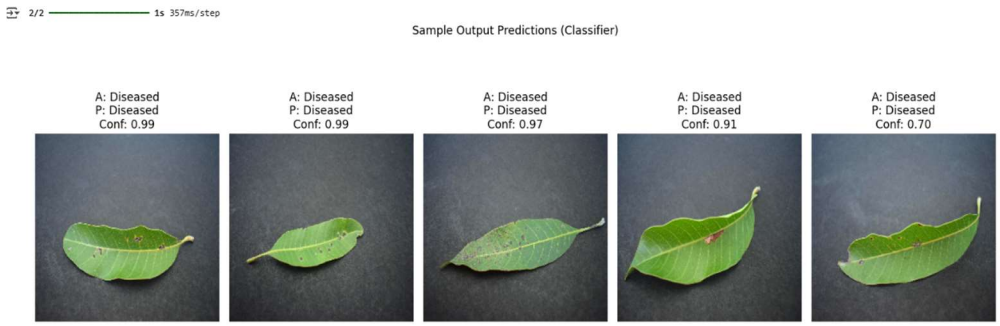
2. **ROC Curve with AUC Score**

- AUC value of 0.97 shows strong model discrimination.
- Curve moves sharply toward the top-left, indicating high true positive rate.

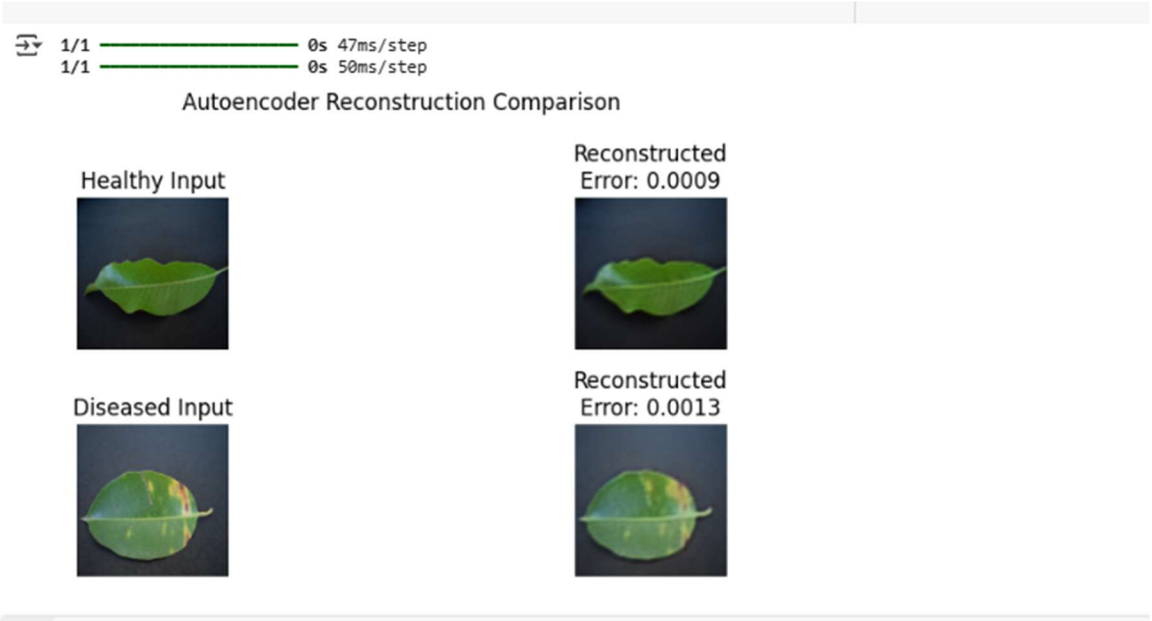


3. **Sample Output Predictions**

- Displays test leaf images with actual vs predicted labels and model confidence.
- Highlights how the model behaves on real data.



4. **Reconstruction Error Comparison** (Autoencoder)



4.3 Downstream Application Integration

4.3.1 Demonstration of Integration with a Downstream Application

To simulate a real-world deployment scenario, the trained classifier and autoencoder were integrated into a single inference pipeline. This combined function takes a new leaf image as input and performs two tasks:

1. **Classification** using the MobileNetV2-based model to predict whether the leaf is healthy or diseased.
2. **Anomaly detection** using a convolutional autoencoder trained only on healthy leaf images. The autoencoder computes reconstruction error, helping to validate or challenge the classifier's prediction.

This integration mimics how the model could be embedded into a mobile or web application for **agriculture specialists or farmers**, enabling them to assess plant health with a single uploaded photo.

4.3.2 Providing Actionable Insights

To demonstrate the practical utility of the developed machine learning system, I have extended the solution to process and classify frames from video input. This simulates a real-world deployment scenario where a drone, surveillance camera, or mobile device continuously captures leaf footage in an agricultural setting.

The integrated pipeline processes individual frames and overlays the following insights in real time:

- **Predicted health status** using the MobileNetV2-based classifier
- **Confidence score** for each prediction
- **Reconstruction error** calculated by the autoencoder
- **Anomaly status**, based on a defined reconstruction error threshold

This system enables real-time visual monitoring and decision-making for farm workers or agronomists. By reviewing the annotated frames, users can quickly identify problematic plants and respond accordingly — making it ideal for **precision agriculture** workflows.

Frame 1: Diseased (1.00) | Error: 0.01147 [Anomaly]

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step

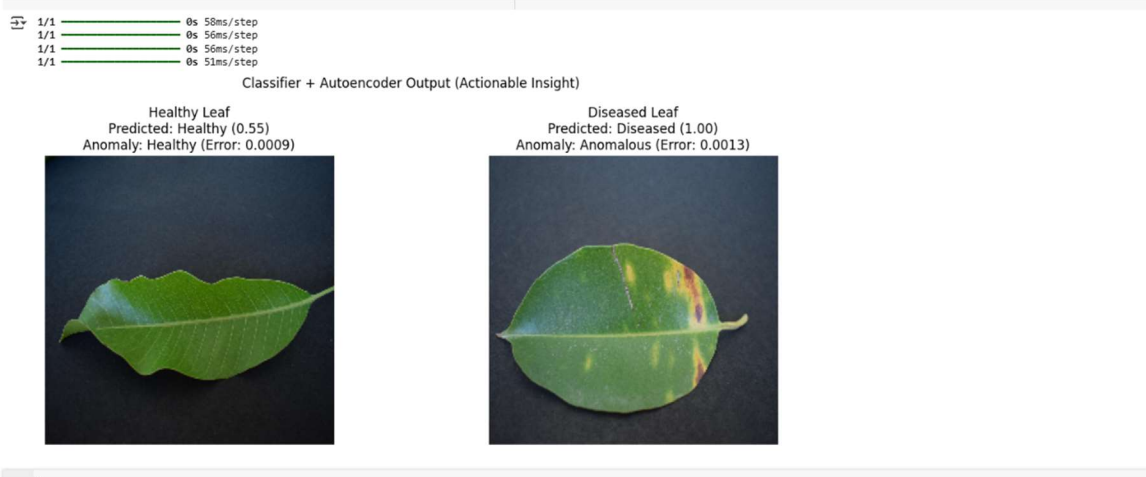
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step

Frame 2: Diseased (0.99) | Error: 0.01166 [Anomaly]

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step

The final integrated function returns a structured output including:



- The **predicted label** (Healthy or Diseased)
- **Confidence score** from the classifier
- **Reconstruction error** from the autoencoder
- **Autoencoder anomaly status** (Normal or Anomalous)

This information provides **actionable insights**:

- If both models agree the leaf is diseased, the farmer is advised to inspect the plant or apply treatment.
- If the classifier says "healthy" but the autoencoder shows high error, this could signal a borderline or rare condition — encouraging further inspection.
- If both models agree on "healthy", the leaf is considered safe.

This dual-validation system increases reliability and minimizes false negatives, which is crucial in **early detection of plant diseases**.

5. Discussion and Conclusion

5.1. Interpretation of Results

The combined classifier and autoencoder approach performed effectively, achieving high test accuracy and strong ROC-AUC.
Confusion matrix revealed a low misclassification rate, especially for healthy leaves.
Sample predictions demonstrated the model's ability to distinguish leaf types with confidence.
Reconstruction error clearly separated healthy from diseased leaves using the autoencoder.

The integrated inference function successfully combined classification and anomaly detection to support real-world decisions.

5.2. Limitations and Future Work

5.2.1. Limitations

- Dataset had minor class imbalance despite mitigation efforts.
- Autoencoder was trained only on limited clases, limiting exposure to edge cases.
- Real-world variability (e.g., lighting, background) may affect prediction quality.

5.2.2. Future Work

- Augment dataset with more diverse disease types and field images.
- Improve generalization by training autoencoder on broader examples.
- Deploy the solution to a user-friendly web or mobile app for farmers.