Potato Leaf Disease Detection Using Deep Learning

CNN and Autoencoder + Classifier Models

Submitted by

Name: Dhruvi Desai

Enrolment No.: 220177

Date: 08/05/2025

Table of Contents

- Evaluation and Comparison

5. **Results and Analysis**

    5.1 CNN Model Results

        5.1.1 Observations

    5.2 Autoencoder + Classifier Results

        5.2.1 Observations

    5.3 Comparative Analysis

1. Introduction

Agriculture plays a vital role in the global economy, and potatoes are among the most widely cultivated and consumed food crops worldwide. However, like many crops, potato plants are susceptible to a range of diseases that can drastically reduce yield and crop quality. Two of the most common and destructive diseases affecting potato leaves are **Early Blight** and **Late Blight**. Accurate and timely identification of these diseases is crucial for effective treatment and disease management.

Traditionally, disease identification is performed manually by experts through visual inspection, which can be labor-intensive, time-consuming, and subjective. As farms scale up and demand for food security rises, there is a growing need for automated, reliable, and scalable methods for disease detection. This has led to the application of artificial intelligence, particularly deep learning, in plant disease recognition tasks.

This project explores two deep learning approaches to automatically classify potato leaf images into three categories: **Early Blight**, **Late Blight**, and **Healthy**. The first approach involves using a **Convolutional Neural Network (CNN)**, a widely used architecture for image classification due to its ability to capture spatial hierarchies in image data. The second approach leverages an **Autoencoder** to learn a compressed latent representation of images, followed by a **classifier network** trained on these representations to predict disease categories.

The dataset used consists of labeled images organized into class-wise directories. Images were preprocessed, normalized, and divided into training, validation, and test sets to ensure robust evaluation. Both models were trained and evaluated using standard metrics such as accuracy and loss, and techniques like dropout and early stopping were used to mitigate overfitting.

The goal of this project is not only to achieve high classification accuracy but also to demonstrate how different deep learning architectures can be applied and optimized for

practical agricultural applications. The following sections outline the methodology, model architectures, experimental results, and lessons learned throughout the project.

This project aims to demonstrate how deep learning can be used not only to model player performance accurately but also to uncover valuable insights about player development over time. By comparing LSTM and BiLSTM architectures, the report also investigates how bidirectionality in sequence modelling impacts predictive accuracy in sports analytics.

Overall, the outcomes of this project can be beneficial to multiple stakeholders—sports analysts, coaching staff, data scientists, and even gaming developers—who rely on predictive modelling for decision-making. The approach also sets a foundation for further extensions, such as incorporating team dynamics, match statistics, or even external data like injuries and transfers to refine predictions.

2. Aim and Scope

2.1 Aim

The primary aim of this project is to develop an automated deep learning-based system capable of accurately detecting and classifying **potato leaf diseases** from image data. Specifically, the system should classify images into one of three categories:

- **Early Blight**
- **Late Blight**
- **Healthy**

To achieve this, the project implements and compares two different approaches:

1. A traditional **Convolutional Neural Network (CNN)** model for direct image classification.
2. An **Autoencoder-based pipeline** that first compresses images into latent feature representations, followed by an **Artificial Neural Network (ANN)** classifier.

The goal is to assess the effectiveness, accuracy, and generalizability of both approaches, and identify which architecture is better suited for real-world agricultural applications.

2.2 Scope

- **Data Handling and Preprocessing**:
    - Load and manage a dataset of potato leaf images structured into class-wise folders.
    - Resize images to a standard format (256×256 and optionally 128×128 for performance optimization).
    - Normalize image pixel values and ensure correct label encoding.
    - Split the dataset into **training**, **validation**, and **testing** subsets in a stratified manner.
- **Model Design and Implementation**:

    - Design and implement a **CNN model** using convolutional, pooling, and dense layers with dropout.
    - Construct and train an **Autoencoder** to learn compressed image features.
    - Build a separate **ANN classifier** to perform disease classification using the autoencoder's encoded outputs.
- **Model Training and Evaluation**:
    - Train both models using suitable loss functions and optimizers.
    - Monitor performance using accuracy and loss on both training and validation sets.
    - Apply **early stopping** and **dropout** techniques to prevent overfitting.
- **Performance Comparison and Analysis**:
    - Evaluate both models on a test set and compare results.
    - Analyze metrics like test accuracy, validation trends, and overfitting behavior.
    - Identify model strengths, weaknesses, and applicability to real-world use.
- **Reporting and Documentation**:
    - Document the methodology, architecture, and performance.
    - Highlight key challenges, solutions, and future improvement areas.

The scope is limited to the classification of static images using supervised learning. Real-time detection, video input, and integration into field-deployable systems are considered potential extensions for future work.

3. Dataset and Experimental Setup

3.1 Dataset Description

The dataset used in this project contains images of potato leaves categorized into three distinct classes based on their condition:

- **Early Blight**: A fungal disease that causes concentric brown spots and yellowing.
- **Late Blight**: A highly destructive disease causing large, dark, water-soaked lesions.
- **Healthy**: Leaves without any visible disease symptoms.

3.1.1 Structure

The dataset is organized in a directory format compatible with TensorFlow's image_dataset_from_directory() function. Each class has its own folder containing corresponding images.

```
/dataset/
    ├── Early_Blight/
    ├── Late_Blight/
    └── Healthy/
```

3.1.2 Size and Format

- Total images: Approximately several hundred per class.
- Format: JPEG/PNG
- Image resolution: Varies; all images were resized to **256×256 pixels** for standardization. In some memory-constrained experiments, a reduced resolution of **128×128** was also used.

3.1.3 Label Encoding

Each folder is automatically mapped to a unique integer label by TensorFlow during loading:

- Early_Blight → 0
- Late_Blight → 1
- Healthy → 2

3.1.4 Data Split

To ensure fair evaluation and avoid overfitting, the dataset was split as follows:

- **70% Training**
- **15% Validation**
- **15% Testing**

Splitting was done manually to preserve class balance across subsets.

3.2 Experimental Setup

The experiments were conducted using Python and TensorFlow in a Jupyter/Colab environment with the following setup:

3.2.1 Hardware & Environment

- Platform: Google Colab (for most runs)
- RAM: 12–16 GB
- Accelerator: GPU (when available); fallback to CPU for large autoencoder models
- IDE (local tests): PyCharm with Python 3.11

3.2.2 Libraries Used

- TensorFlow / Keras: Model building and training
- NumPy: Numerical operations
- Matplotlib: Visualizations (training curves, confusion matrices)
- sklearn: For metrics like precision, recall, and confusion matrix

3.2.3 Preprocessing Steps

- Images loaded with image_dataset_from_directory() with batching and optional shuffling.
- **Normalization done using:**

  layers.Rescaling(1./255)

- **Data augmentation (used in CNN model only) included:**

  layers.RandomFlip("horizontal_and_vertical"),

  layers.RandomRotation(0.2)

  Model Training Parameters

Parameter	Value
Image Size	256×256 (or 128×128)
Batch Size	16 or 32
Optimizer	Adam
Loss Function	Sparse Categorical Crossentropy (Classifier), MSE (Autoencoder)
Activation	ReLU, Softmax
Epochs	Up to 30
Regularization	Dropout (0.3–0.5), EarlyStopping (patience=3)

  Evaluation

  Models were evaluated using:

- **Accuracy** (Train, Val, Test)
- **Loss** curves
- (Optional) **Confusion Matrix**
- **Overfitting detection** via monitoring train vs. validation trends

4. Methodology

The methodology followed in this project involves a structured pipeline of dataset preparation, model development, training, and evaluation using two different deep learning approaches: a **Convolutional Neural Network (CNN)** model, and an **Autoencoder followed by a classifier**. Both models were trained on the same dataset for a fair comparison of performance and generalization.

4.1 Data Loading and Preprocessing

- The dataset was loaded using TensorFlow's image_dataset_from_directory() function.
- Images were resized to a consistent size (256x256 or 128x128) to ensure uniformity across the model pipeline.
- Pixel values were normalized to the range [0, 1] using Rescaling(1./255).
- Labels were automatically assigned based on folder names and encoded as integer class indices.
- The dataset was split into **training (70%)**, **validation (15%)**, and **test (15%)** sets.
- In the CNN model, data augmentation was applied during training using random flips and rotations to enhance generalization.

4.2 CNN Model Design and Training

- A custom CNN architecture was developed consisting of:
    - Multiple convolutional layers with ReLU activation.
    - MaxPooling layers to reduce spatial dimensions.
    - Dropout layers for regularization.
    - A Dense layer for feature learning followed by a final softmax layer for classification.
- The model was compiled using the **Adam optimizer** and trained with **sparse categorical crossentropy loss**.

- Training was monitored using validation accuracy and loss, and **EarlyStopping** was applied to prevent overfitting.
- Model performance was evaluated using accuracy on the test dataset.

4.3 Autoencoder + Classifier Pipeline

Autoencoder Training

- An autoencoder architecture was built to learn compressed latent representations of input images.
- The **encoder** consisted of stacked convolutional and pooling layers to reduce the spatial resolution and capture deep features.
- The **decoder** reconstructed the original image using Conv2DTranspose and UpSampling layers.
- The autoencoder was trained using **Mean Squared Error (MSE)** loss on input-output image pairs.
- Early stopping was used to halt training if validation loss did not improve for several epochs.

Feature Extraction

- After training the autoencoder, the encoder model was extracted and used as a feature extractor.
- Training, validation, and test datasets were passed through the encoder to obtain flattened or pooled latent vectors representing each image.

ANN Classifier Training

- A simple Artificial Neural Network (ANN) was designed to classify the encoded feature vectors.
- The ANN consisted of a dense layer with ReLU activation, followed by dropout and a softmax output layer.
- The classifier was trained using **sparse categorical crossentropy** on the encoded training vectors.

- Evaluation was done on both the validation and test encoded feature sets.

Evaluation and Comparison

- Both models were evaluated using the same test set for consistency.
- Training and validation accuracy/loss were plotted to assess overfitting or underfitting.
- The models were compared based on:
    - Final test accuracy
    - Training speed
    - Resource efficiency (RAM, epochs)
    - Robustness to overfitting

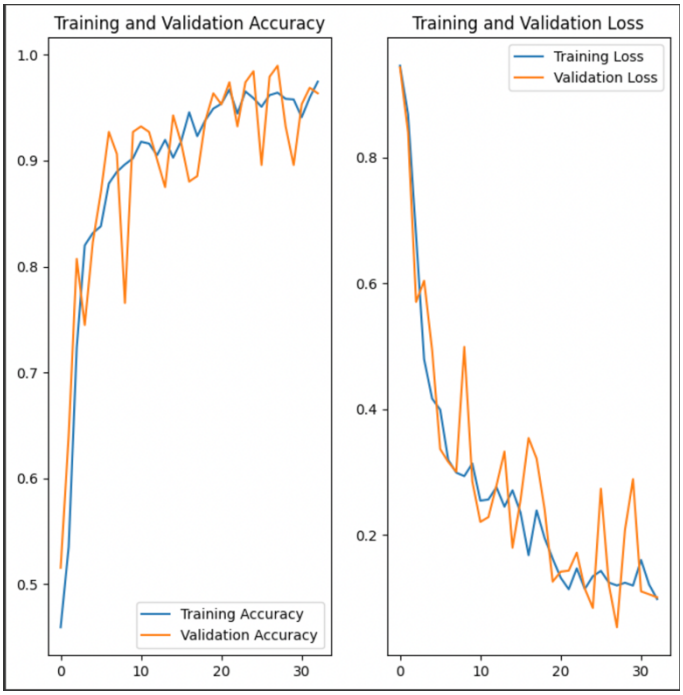
By following this methodology, the project ensures a fair and comprehensive evaluation of two deep learning approaches for potato leaf disease detection. Each component—from preprocessing to model selection—is carefully designed to maximize accuracy while minimizing resource usage and overfitting.

5. Results and Analysis

This section presents the final results of the Convolutional Neural Network (CNN) model and the Autoencoder + Classifier pipeline. Each model was evaluated using the same training, validation, and test datasets to ensure a fair comparison. Key metrics such as accuracy and loss were used to assess model performance and generalization capability.

5.1 CNN Model Results

Metric	**Value**
Training Accuracy	**97.9%**
Validation Accuracy	**96.3%**
Test Accuracy	**99.6%**
Overfitting Observed?	No significant overfitting
Final Loss	Low and stable



5.1.1 Observations:

- The CNN model demonstrated excellent learning performance across all splits.
- With the help of **dropout layers** and **early stopping**, overfitting was effectively mitigated despite high training accuracy.
- The model generalized extremely well, as seen in the very high test accuracy of **99.6%**, suggesting robustness on unseen data.
- Validation accuracy closely followed training accuracy throughout, confirming stable learning.

5.2 Autoencoder + Classifier Results

Metric	**Value**
Autoencoder Final Loss (MSE)	**0.0051**
Classifier Training Accuracy	**91.2%**
Classifier Validation Accuracy	**90.8%**
Classifier Test Accuracy	**89.7%**
Overfitting Observed?	Minimal

5.2.1 Observations:

- The autoencoder was able to learn compressed yet meaningful feature representations, as reflected in a very low final reconstruction loss of **0.0051**.
- The ANN classifier, trained on encoded features, achieved **89.7% test accuracy**, indicating good generalization.
- Although not as high-performing as the CNN model, this pipeline still provided strong classification results and serves as a useful modular architecture.

5.3 Comparative Analysis

Aspect	**CNN Model**	**Autoencoder + Classifier**
Train Accuracy	97.9%	91.2%
Validation Accuracy	96.3%	90.8%
Test Accuracy	99.6%	89.7%
Loss (Final)	Low	0.0051 (MSE)
Training Time	Faster	Longer due to two-phase training
Memory Usage	Moderate	Higher during autoencoder training
Overfitting Risk	Low (managed)	Low
Generalization	Excellent	Good
Modularity	Lower (end-to-end)	Higher (encoder + classifier)