Plant Disease Classification with CNN

Duran Duman 190101037

April 2024

Abstract

This project aims to solve the problem of identifying plant diseases by using Convolutional Neural Networks (CNNs). Focusing on potato diseases, the project creates a deep learning model that can classify images of potato leaves into three categories: healthy, early blight, and late blight. The model is trained with a large dataset of thousands of labeled images, which helps it learn the detailed patterns of each disease. The implementation uses Python and powerful deep learning libraries like TensorFlow and Keras. The main goal is to improve agricultural productivity by providing an automated tool for early and accurate disease detection. This will help farmers and agricultural professionals manage crops and control diseases more effectively. This report explains the methodology, dataset, model architecture, training process, and performance evaluation. It highlights the importance of CNNs in detecting plant diseases and discusses the wider impact on food security and sustainable agriculture.

1 Introduction

Before we delve into the specifics of the ”Plant Disease Classification with CNN” project, let’s get fa- miliar with Convolutional Neural Networks (CNNs) and where they come in handy.

Convolutional Neural Networks (CNNs) have revolutionized the field of image recognition and classification, making significant strides in various domains, including agriculture. This project, ”Plant Disease Classification with CNN,” leverages CNNs to tackle a pressing issue in agriculture: the detection and classification of plant diseases. CNNs are a specialized type of artificial neural network designed to process and analyze visual data by automatically detecting important features and patterns using layers of convolutional filters. Their ability to handle the complex structure of image data makes them particularly suitable for identifying diseases in plants, which often present subtle visual symptoms.

1.1 Areas of Use

CNNs find applications in several areas, demonstrating their versatility and effectiveness [6] :

- **Computer Vision:** CNNs are extensively used for image classification, object detection, segmentation, and recognition. They enable machines to interpret and understand visual data, playing a crucial role in applications such as autonomous vehicles, medical imaging, surveillance systems, and augmented reality.
- **Natural Language Processing (NLP):** Although primarily used for images, CNNs are also adapted for text tasks in NLP, such as text classification and sentiment analysis.
- **Medical Imaging:** In healthcare, CNNs assist in diagnosing diseases from medical images like X-rays and MRI scans, helping doctors identify tumors and other abnormalities.
- **Video Analysis:** CNNs are employed to interpret video data, facilitating tasks like action recognition, video summarization, and activity detection in surveillance footage.
- **Remote Sensing:** CNNs analyze satellite images to monitor crops, manage natural disasters, and track environmental changes.
- **Robotics:** In robotics, CNNs help robots understand their environment through cameras, aiding in tasks like object manipulation, navigation, and human interaction.

1.2 Advantages of CNNs

CNNs offer several key advantages that make them suitable for image-based tasks [7] :

- **Automatic Feature Learning:** CNNs automatically learn relevant features from data, eliminating the need for manual feature extraction.
- **Translation Invariance:** They can recognize patterns regardless of their position in the image.
- **Hierarchical Structure:** CNNs build a hierarchical representation of data, capturing complex patterns at multiple levels of abstraction.
- **Parameter Sharing:** CNNs use parameter sharing, reducing the number of parameters and improving generalization.
- **Efficiency with Large Images:** CNNs efficiently handle large images due to their local connectivity, which limits the number of connections between layers.

2 Problem Description

Plant diseases pose a significant threat to agricultural productivity, leading to considerable economic losses and food security concerns. Traditional methods of disease detection often rely on manual inspection by experts, which can be time-consuming, and prone to errors.

The primary objective of this project is to develop an automated system for detecting plant diseases, specifically targeting potato plants. By employing CNNs, the project aims to create a model that can accurately classify images of potato leaves into three categories: healthy, early blight, and late blight.

Potato plants are particularly vulnerable to early blight and late blight, which can cause substantial damage if not identified and managed promptly. Early blight, caused by the fungus *Alternaria solani*, manifests as dark spots with concentric rings on the leaves. Late blight, caused by the oomycete *Phytophthora infestans*, is characterized by large, irregularly shaped lesions that can quickly spread across the plant [8]. Accurate and timely identification of these diseases is crucial for effective intervention and treatment.

3 Libraries Used

This project leverages several powerful libraries in Python to implement the CNN model and visualize the results:

- **TensorFlow:** TensorFlow is an open-source deep learning framework developed by Google. It provides a comprehensive ecosystem for building and deploying machine learning models. TensorFlow offers tools for constructing neural networks, optimizing training processes, and deploying models in production environments. Its flexibility and scalability make it suitable for a wide range of machine learning tasks.[1]

- **Keras:** Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It simplifies the process of building and training deep learning models by providing user-friendly functions and abstractions. Keras is particularly known for its ease of use, which allows researchers and developers to quickly prototype and experiment with different neural network architectures.[2]

- **Matplotlib:** Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications. In this project, Matplotlib is used to visualize the training and validation accuracy and loss, as well as to display sample images from the dataset.[3]

- **NumPy:** NumPy is a fundamental package for scientific computing with Python. It provides support for arrays, matrices, and a wide range of mathematical functions. NumPy's powerful n-dimensional array object is utilized extensively in data preprocessing and manipulation tasks, which are essential for preparing the dataset for training the CNN model.[4]

- **Seaborn (sns):** Seaborn is a data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn is used in this project to generate detailed and aesthetically pleasing visualizations, such as confusion matrices, to better understand the model's performance.[5]

4 Dataset Overview

4.1 Dataset Name: New Plant Disease Dataset

The dataset used in this project is the New Plant Disease Dataset, sourced from Kaggle. It comprises approximately 87,000 RGB images of healthy and diseased plant leaves, divided into 38 different classes. For the purposes of this project, only images pertaining to potato diseases were utilized. The potato dataset includes three classes: "potato early blight," "late blight," and "healthy." The dataset was split into 5632 training images and 1440 validation images, with each image sized at 256x256 pixels.

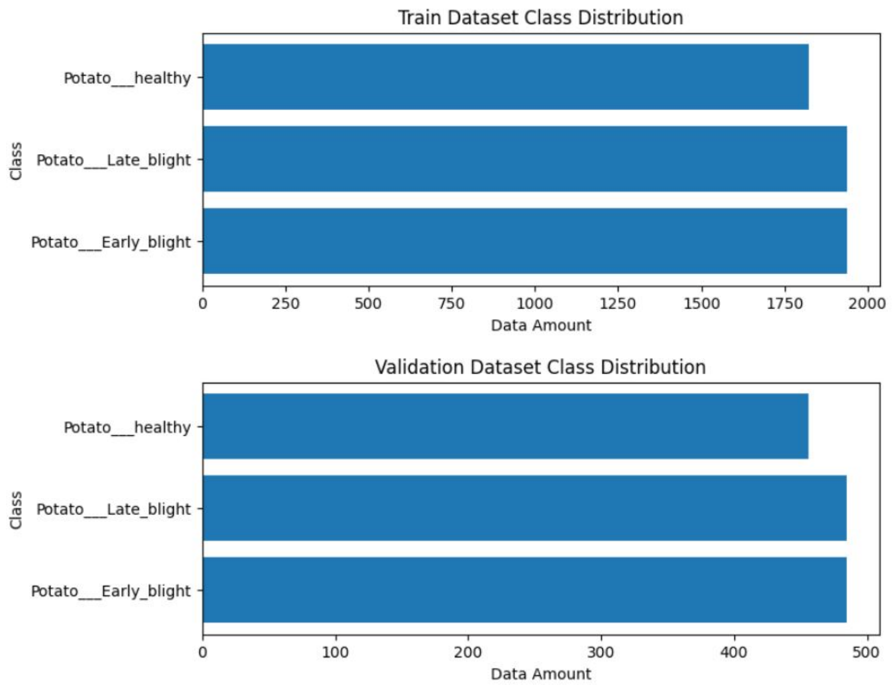


Figure 1: Distributon of Classes of Train and Validation Datasets

4.2 Data Visualization

To gain an understanding of the dataset, sample images from each class were visualized. This step involved displaying images labeled as early blight, late blight, and healthy to observe the distinct visual characteristics of each category.

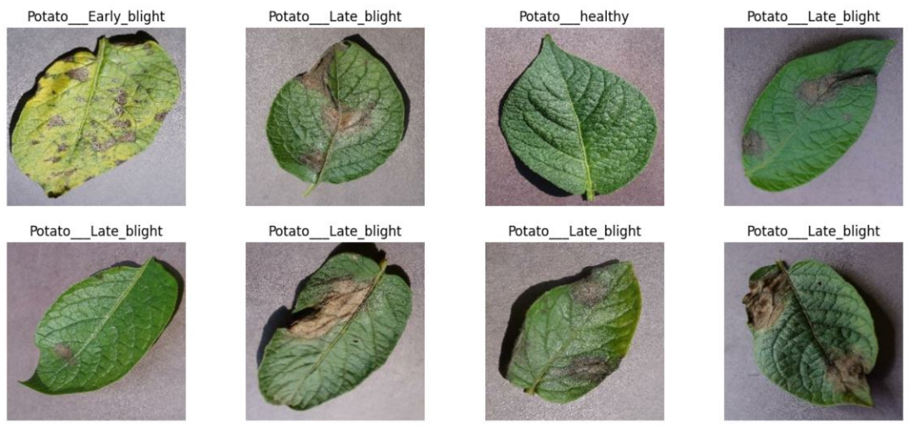
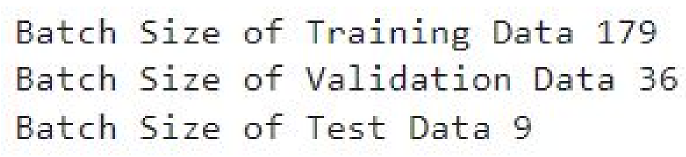


Figure 2: Sample Images from the Dataset

4.3 Splitting the Validation Dataset

The validation dataset was further partitioned to create a separate test set for evaluating the model's performance. The validation set comprised 80% of the original validation dataset, while the remaining 20% formed the test set. This partitioning resulted in 36 batches for validation data and 9 batches for test data, with a batch size of 32. Batches refer to subsets of the dataset processed together during each iteration of training, facilitating efficient and effective model learning.

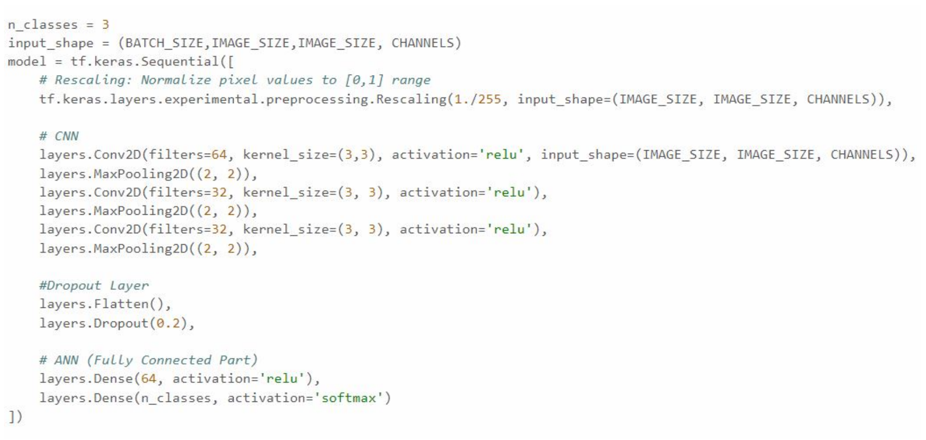


```
Batch Size of Training Data 179
Batch Size of Validation Data 36
Batch Size of Test Data 9
```

Figure 3: Batch Size of Datasets

5 Model Architecture

The CNN model was meticulously designed to classify potato leaf images into the three disease categories. The architecture included the following layers:



```
n_classes = 3
input_shape = (BATCH_SIZE,IMAGE_SIZE,IMAGE_SIZE, CHANNELS)
model = tf.keras.Sequential([
    # Rescaling: Normalize pixel values to [0,1] range
    tf.keras.layers.experimental.preprocessing.Rescaling(1./255, input_shape=(IMAGE_SIZE, IMAGE_SIZE, CHANNELS)),

    # CNN
    layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu', input_shape=(IMAGE_SIZE, IMAGE_SIZE, CHANNELS)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    #Dropout Layer
    layers.Flatten(),
    layers.Dropout(0.2),

    # ANN (Fully Connected Part)
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax')
])
```

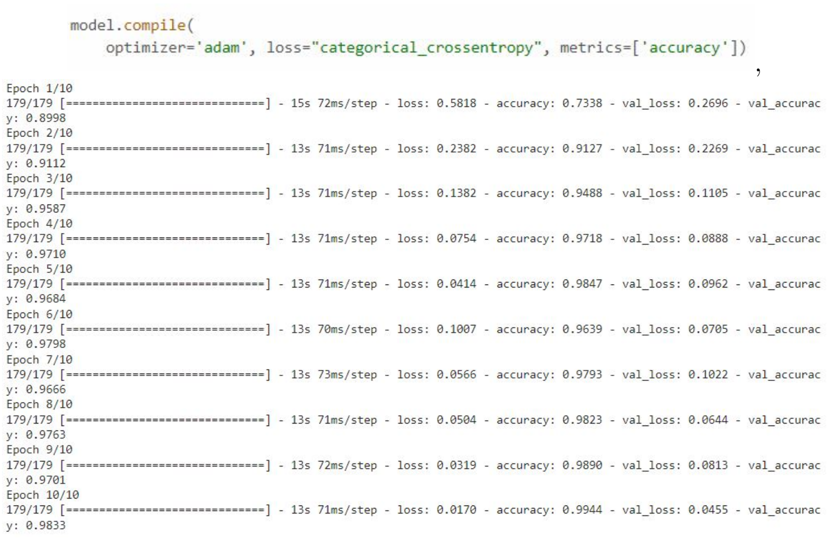
Figure 4: Model Architecture

1. **Input Layer:**
   - **Rescaling Layer:** Normalizes pixel values to the range [0, 1], ensuring consistent input by scaling pixel values from 0-255 to a normalized range suitable for training.
2. **Convolutional Layers:**
   - **First Convolutional Layer:** Uses 64 filters with a size of 3x3, applying ReLU (Rectified Linear Unit) activation function. This layer detects low-level features such as edges and textures.
   - **Second Convolutional Layer:** Uses 32 filters with a size of 3x3 and ReLU activation, capturing more complex patterns by building on features detected by the first layer.
   - **Third Convolutional Layer:** Uses 32 filters with a size of 3x3 and ReLU activation, extracting even higher-level features from the images.
3. **MaxPooling Layers:**
   - **First MaxPooling Layer:** Uses a pool size of 2x2, reducing the dimensionality of feature maps and retaining essential information while reducing computational complexity.
   - **Second MaxPooling Layer:** Also uses a pool size of 2x2, further downsampling the feature maps to focus on the most relevant features.

4. **Dropout Layer:**
    - **First Dropout Layer:** Set with a dropout rate of 0.2, this layer randomly drops neurons during training to prevent overfitting and promote generalization.
5. **Fully Connected Layers:**
    - **First Fully Connected (Dense) Layer:** Comprises 64 neurons with ReLU activation, integrating features extracted by convolutional layers to perform the final classification.
6. **Output Layer:**
    - **Dense Layer:** This layer has three neurons corresponding to the three classes (healthy, early blight, late blight) with a softmax activation function to output probabilities for each class.

5.1 Model Compilation and Training

The model was compiled using the Adam optimizer, which adapts the learning rate during training to optimize performance. The categorical cross-entropy loss function was employed to measure the difference between predicted and actual class labels, and accuracy metrics were used to evaluate the model's performance. The training process involved 10 epochs, with each epoch representing one complete pass through the training dataset.



```
model.compile(
    optimizer='adam', loss="categorical_crossentropy", metrics=['accuracy'])
```
,

```
Epoch 1/10
179/179 [==============================] - 15s 72ms/step - loss: 0.5818 - accuracy: 0.7338 - val_loss: 0.2696 - val_accurac
y: 0.8998
Epoch 2/10
179/179 [==============================] - 13s 71ms/step - loss: 0.2382 - accuracy: 0.9127 - val_loss: 0.2269 - val_accurac
y: 0.9112
Epoch 3/10
179/179 [==============================] - 13s 71ms/step - loss: 0.1382 - accuracy: 0.9488 - val_loss: 0.1105 - val_accurac
y: 0.9587
Epoch 4/10
179/179 [==============================] - 13s 71ms/step - loss: 0.0754 - accuracy: 0.9718 - val_loss: 0.0888 - val_accurac
y: 0.9710
Epoch 5/10
179/179 [==============================] - 13s 71ms/step - loss: 0.0414 - accuracy: 0.9847 - val_loss: 0.0962 - val_accurac
y: 0.9684
Epoch 6/10
179/179 [==============================] - 13s 70ms/step - loss: 0.1007 - accuracy: 0.9639 - val_loss: 0.0705 - val_accurac
y: 0.9798
Epoch 7/10
179/179 [==============================] - 13s 73ms/step - loss: 0.0566 - accuracy: 0.9793 - val_loss: 0.1022 - val_accurac
y: 0.9666
Epoch 8/10
179/179 [==============================] - 13s 71ms/step - loss: 0.0504 - accuracy: 0.9823 - val_loss: 0.0644 - val_accurac
y: 0.9763
Epoch 9/10
179/179 [==============================] - 13s 72ms/step - loss: 0.0319 - accuracy: 0.9890 - val_loss: 0.0813 - val_accurac
y: 0.9701
Epoch 10/10
179/179 [==============================] - 13s 71ms/step - loss: 0.0170 - accuracy: 0.9944 - val_loss: 0.0455 - val_accurac
y: 0.9833
```

Figure 5: Model's Compile Parameters and Accuracy Metrics

5.2 Model Evaluation

Graphs depicting training and validation accuracy, as well as loss, were plotted to visualize the model's performance over epochs. The final evaluation of the model was conducted using the test dataset, achieving an accuracy of 98.96%. A confusion matrix was generated to visualize the classification results across different classes, providing insight into the model's precision and recall.

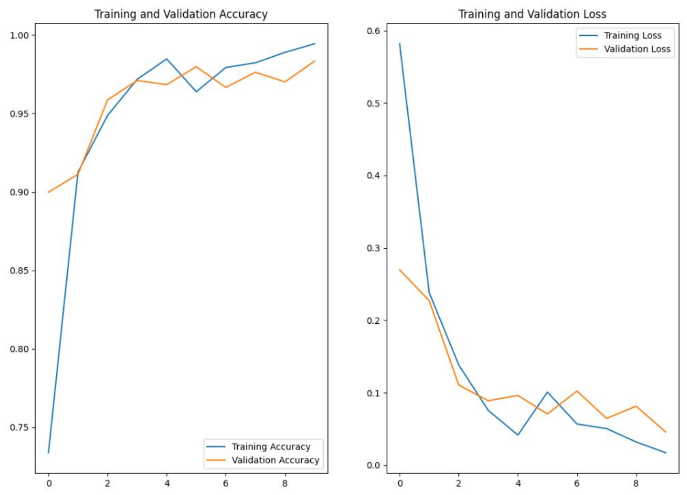


Figure 6: Accuracy, Loss Diagram of Train and Validation Datasets

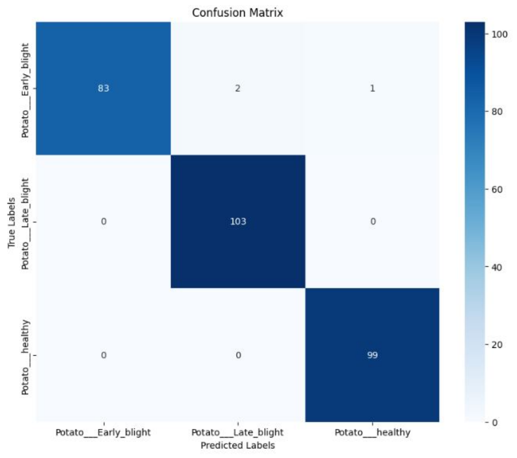
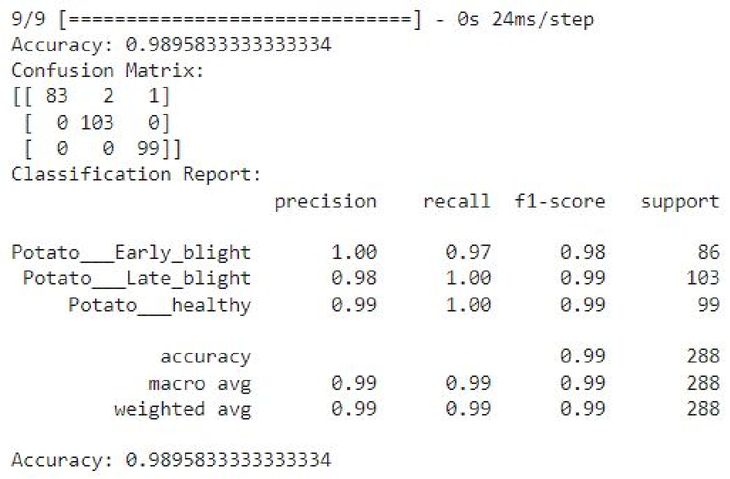


Figure 7: Confusion Matrix of Test Data

5.3 Performance Metrics

Performance metrics are essential to evaluate the effectiveness and reliability of the CNN model. Here, we consider precision, recall, and F1-score for each class, alongside overall accuracy. These metrics provide a understanding of the model's performance in identifying and distinguishing between healthy, early blight, and late blight potato leaves.

- **Precision:** Precision measures the accuracy of the positive predictions made by the model. It is calculated as (TP / (TP +FP)), where TP is the number of true positives and FP is the number of false positives. High precision indicates that the model has a low false positive rate.
- **Recall:** Recall, or sensitivity, measures the model's ability to correctly identify all actual positive cases. It is calculated as (TP / (TP +FN)), where FN is the number of false negatives. High recall means the model can identify most of the actual positive cases.
- **F1-Score:** The F1-score is the harmonic mean of precision and recall, calculated as (2* precision * recall) / (precision + recall). It provides a balanced measure of the model's performance.



```
9/9 [==============================] - 0s 24ms/step
Accuracy: 0.9895833333333334
Confusion Matrix:
[[ 83   2   1]
 [  0 103   0]
 [  0   0  99]]
Classification Report:
                       precision    recall  f1-score   support

Potato___Early_blight       1.00      0.97      0.98        86
 Potato___Late_blight       0.98      1.00      0.99       103
     Potato___healthy       0.99      1.00      0.99        99

             accuracy                           0.99       288
            macro avg       0.99      0.99      0.99       288
         weighted avg       0.99      0.99      0.99       288

Accuracy: 0.9895833333333334
```

Figure 8: Precision, Recall, and F1-Score for Each Class

The classification report (Figure 8) highlights the following key points:

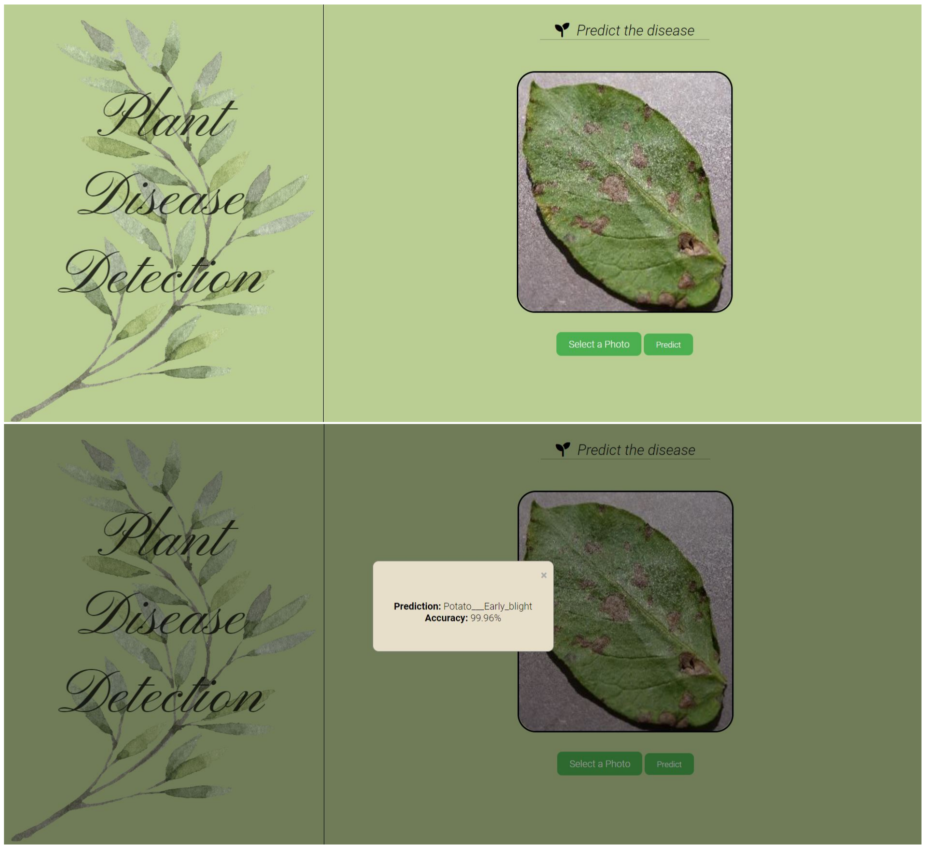
- **Healthy Class:** Achieved a precision of 99, recall of 100, and F1-score of 99 out of 100, indicating excellent performance in correctly identifying healthy potato leaves with minimal false positives and negatives.
- **Early Blight Class:** Achieved a precision of 100, recall of 97, and F1-score of 98 out of 100. The high recall shows that the model is highly effective in detecting early blight, which is crucial for timely intervention.
- **Late Blight Class:** Achieved a precision of 98, recall of 100, and F1-score of 99 out of 100. This near-perfect performance underscores the model's capability to accurately identify late blight, a critical factor in preventing damage.

Overall, the model demonstrated an impressive accuracy of 98.96 on the test dataset, confirming its robustness and reliability. These metrics indicate that the CNN model is highly effective in classifying potato leaf images across all three categories. The high precision and recall values suggest that the model can be trusted to make accurate diagnoses.

6 Innovation - 'Webpage with Flask Backend'

To extend the usability and accessibility of the developed CNN model, a web application was created using the Flask framework. This web-based tool provides an intuitive interface for users to upload images of potato leaves and receive real-time disease classification results.

Github Repository of The Project:
https://github.com/duranduman06/Plant-Disease-Classification-Using-CNN



Prediction: Potato___Early_blight
Accuracy: 99.96%

Figure 9: Prediction of Selected Potato Image's Class and Accuracy Value

6.1 Key Features of the Web Application

- **User-Friendly Interface:** The web application features a simple and clean interface where users can easily upload images of potato leaves. The design ensures that everybody can use the tool effectively.

- **Real-Time Predictions:** Once an image is uploaded, the application processes it through the CNN model and provides instant predictions. This feature is crucial for farmers and agricultural professionals who need quick and accurate disease diagnoses.

- **Detailed Results:** The application not only displays the predicted class (healthy, early blight, or late blight) but also provides the confidence score of the prediction. This helps users understand the reliability of the results and make decisions.

6.2 Technical Implementation

The web application was developed using Flask, a web application framework in Python. Flask is chosen for its simplicity and flexibility, allowing for rapid development and easy integration with the CNN model. The application architecture includes the following components:

- **Frontend:** Implemented using HTML, CSS, and JavaScript to create an interactive user interface.

- **Backend:** The Flask backend handles user requests, processes uploaded images, and interacts with the trained CNN model to generate predictions. TensorFlow and Keras libraries are utilized to load the pre-trained model and perform inference on the uploaded images.

Future work could involve expanding the application to include more plant species and diseases, incorporating advanced image preprocessing techniques to handle varied lighting conditions and backgrounds, and integrating additional features such as disease treatment recommendations.

In conclusion, the development of a web application using Flask for plant disease classification demonstrates a practical application of CNNs in agriculture, showcasing how advanced AI techniques can be translated into user-friendly tools with significant real-world impact.

7 Conclusion

This project successfully developed a Convolutional Neural Network (CNN) model that can accurately classify potato plant diseases into three categories: healthy, early blight, and late blight. This result shows the power of deep learning techniques in agriculture, especially for detecting plant diseases. By automating the disease identification process, the CNN model offers a significant advantage over traditional inspection methods. It provides quick and precise diagnoses, which are crucial for timely intervention and effective crop management.

Additionally, integrating this model into a user-friendly web application using the Flask framework makes it more accessible and easier to use. The web application allows farmers and agricultural professionals to easily upload images and receive real-time classification results, helping them make immediate and informed decisions. This innovation not only improves crop management but also increases agricultural productivity by reducing the need for manual inspections and preventing crop damage.

For future work, there are several directions to explore to extend the impact of this project. These include:

- **Expanding the Model:** Adding more plant species and a wider range of diseases to the model to make it more versatile and applicable in different agricultural contexts.
- **Advanced Image Processing:** Using advanced image preprocessing techniques to improve the model's performance under various environmental conditions, such as different lighting and backgrounds.
- **Additional Features:** Including features like disease treatment recommendations and historical data analysis to provide more comprehensive support for crop management.

In conclusion, the "Plant Disease Classification with CNN" project demonstrates a practical and impactful application of deep learning in agriculture. By turning complex AI models into accessible tools, this project shows the potential for technology to bring significant improvements to agricultural practices, contributing to better food security and sustainable agricultural development.

8 References

1. **https://www.tensorflow.org/about**
2. **https://keras.io/about/**
3. **https://numpy.org/doc/stable/**
4. **https://matplotlib.org/**
5. **https://seaborn.pydata.org/tutorial/introduction.html**
6. **https://medium.com/@AIandInsights/convolutional-neural-networks-cnns-in-computer-vision-10573d0f5b00**
7. **https://medium.com/@khwabkalra1/convolutional-neural-networks-for-image-classification-f0754f7b94aa**
8. **https://ipm.cahnr.uconn.edu/early-blight-and-late-blight-of-potato/**