Visvesvaraya Technological University, Belagavi – 590018

REPORT
ON

CNN Model for Potato Leaf Disease Detection Using TensorFlow

Submitted in partial fulfillment for the award of degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE & ENGINEERING

Submitted by

Shridhara	4SO21CS157
Swastik M Shet	4SO21CS169
T Sharana Sagar	4SO21CS172
Royson DSouza	4SO22CS407

Under the Guidance of

Dr Rio Dsouza
Professor, Department of CSE

DEPT. OF COMPUTER SCIENCE AND ENGINEERING
ST JOSEPH ENGINEERING COLLEGE
An Autonomous Institution
(Affiliated to VTU Belagavi, Recognized by AICTE, Accredited by NBA)
Vamanjoor, Mangaluru - 575028, Karnataka
2023-24

Abstract

Based on the PlantVillage dataset, this study creates and assesses a Convolutional Neural Network (CNN) model for diagnosing potato plant diseases using TensorFlow and Keras. From 2,152 preprocessed and augmented images, the algorithm seeks to properly detect Early Blight, Late Blight, and Healthy states. Six convolutional layers with ReLU activation and max-pooling are part of the CNN architecture. Fully connected layers with a softmax output for multi-class classification come next. After thorough data augmentation and training on an 80-20 dataset, the model successfully distinguished between disease classes and consistently reduced error rates during both training and validation. The model's prediction accuracy is validated by evaluation on an independent test set, underscoring its potential for useful application in agricultural disease control.

To further increase classification accuracy and model generalization, future work may concentrate on improving the model architecture, expanding the dataset, and incorporating more preprocessing methods. The project highlights the importance of deep learning in assisting sustainable farming methods and shows how CNNs may improve agricultural diagnostics.

Contents

Chapter 1

Introduction

Plant diseases have a major negative impact on agricultural productivity, which raises issues with food security and financial losses. Conventional hand examination techniques are laborious and prone to mistakes when it comes to disease identification. The objective of this project is to use TensorFlow and Keras to automatically create a Convolutional Neural Network (CNN) model for the classification of potato plant diseases from the PlantVillage dataset. Images of potato leaves classified as Early Blight, Late Blight, and Healthy are included in the dataset. The model aims to deliver precise and effective disease diagnosis by utilizing deep learning techniques. This would help farmers manage diseases promptly and increase crop yields.

Chapter 2

Architecture of the CNN Model

The architecture of the Convolutional Neural Network (CNN) is designed to effectively capture and learn features from images, enabling accurate classification of potato plant diseases. The model employs a series of convolutional layers, max-pooling operations, and fully connected layers to process the input images, extract meaningful features, and make predictions. Data augmentation and resizing techniques are utilized to enhance the model's generalization capabilities and ensure robust performance across varying conditions.

Input Layer

Resizes and rescales input images to 128x128 pixels and normalizes pixel values.

- **Resizing:** Changes image dimensions to 128x128 pixels.
- **Rescaling:** Scales pixel values to a range of 0 to 1.

Data Augmentation

Increases dataset variability to improve model generalization.

- **Random Flip:** Horizontally and vertically flips images randomly.
- **Random Rotation:** Rotates images randomly within a specified range (0.2 radians).

Convolutional Layers

Extracts features from images using convolutional filters.

- **Conv2D:** Applies convolutional filters (with varying numbers of filters and kernel sizes) to capture different features.
- **Activation:** Uses ReLU activation function to introduce non-linearity.

MaxPooling Layers

Reduces spatial dimensions (width and height) of feature maps while retaining important information.

- **MaxPooling2D:** Applies max pooling with a pool size of 2x2, which downsamples the feature maps.

Flatten Layer

Converts 2D feature maps into a 1D vector to feed into fully connected layers.

- **Flatten:** Reshapes the 2D output into a 1D vector.

Fully Connected (Dense) Layers

Processes the flattened feature vectors and performs classification.

- **Dense:** Fully connected layer with specified units and activation functions.

- **Activation:** Uses ReLU activation for hidden layers and Softmax activation for the output layer to handle multi-class classification.

Layer (type)	**Output Shape**	**No. of Parameter**
Sequential (Resizing, Rescaling)	(None, 128, 128, 3)	0
Sequential (Random Flip, Rotation)	(None, 128, 128, 3)	0
Conv2D (32 filters, 3x3 kernel)	(None, 128, 128, 32)	896
MaxPooling2D (2x2 pool size)	(None, 64, 64, 32)	0
Conv2D (64 filters, 3x3 kernel)	(None, 64, 64, 64)	18496
MaxPooling2D (2x2 pool size)	(None, 32, 32, 64)	0
Conv2D (64 filters, 3x3 kernel)	(None, 32, 32, 64)	36928
MaxPooling2D (2x2 pool size)	(None, 16, 16, 64)	0
Conv2D (64 filters, 3x3 kernel)	(None, 16, 16, 64)	36928
MaxPooling2D (2x2 pool size)	(None, 8, 8, 64)	0
Conv2D (64 filters, 3x3 kernel)	(None, 8, 8, 64)	36928
MaxPooling2D (2x2 pool size)	(None, 4, 4, 64)	0
Conv2D (64 filters, 3x3 kernel)	(None, 4, 4, 64)	36928
MaxPooling2D (2x2 pool size)	(None, 2, 2, 64)	0
Flatten	(None, 256)	0
Dense (64 units, ReLU)	(None, 64)	16448
Dense (3 units, Softmax)	(None, 3)	195

Table 2.1: CNN Model Architecture

Chapter 3

Dataset Description and Splitting

3.1 Dataset Overview

The dataset used for the potato plant disease classification is the PlantVillage dataset. This dataset is specifically designed to address various diseases affecting potato plants and is essential for training and evaluating machine learning models in agricultural diagnostics.

- **Dataset Name:** PlantVillage Dataset
- **Focus:** Potato plant diseases
- **Classes:**
  - Potato_Early_blight
  - Potato_Late_blight
  - Potato_healthy
- **Total Number of Images:** 2,152
- **Image Size:** Each image is resized to 128x128 pixels
- **Batch Size for Training:** 32

3.2 Dataset Splitting

The dataset is divided into three parts to facilitate effective model training and evaluation:

- **Training Set:**
  - **Proportion:** 80% of the dataset
  - **Purpose:** Used to train the model, allowing it to learn the features and patterns associated with different classes of potato plant diseases.
- **Validation Set:**
  - **Proportion:** 20% of the dataset
  - **Purpose:** Used to validate the model during training. It helps in tuning hyperparameters and selecting the best model based on performance metrics without biasing the model to the test data.
- **Test Set:**
  - **Proportion:** Remaining portion after training and validation splits
  - **Purpose:** Used for final evaluation of the model's performance. This set assesses how well the model generalizes to unseen data.

Chapter 4

Implementation

4.1 Training Algorithm and Tools Used

4.1.1 Training Algorithm

The CNN model for potato plant disease classification was trained using the following steps:

- **Data Preprocessing:** Images were resized to 128x128 pixels and pixel values were normalized.
- **Data Augmentation:** Random horizontal and vertical flips, as well as random rotations, were applied to increase dataset variability.
- **Compilation:** The model was compiled using the Adam optimizer and sparse categorical cross-entropy loss function.
- **Training:** The model was trained for 50 epochs using the training dataset, with validation performed on a separate validation set.

4.1.2 Tools Used

- **TensorFlow and Keras:** For building and training the CNN model.

- **Matplotlib:** For visualizing the training and validation metrics.
- **Google Colab:** For executing the training process and managing the dataset.
- **Python:** The programming language used for implementing the model and its training process.

4.1.3 Program Code

```
import tensorflow as tf
from tensorflow.keras import models, layers
import numpy as np

IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/PlantVillage",
    shuffle=True,
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE )

class_names = dataset.class_names
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.2, shuf
    ds_size = len(ds)
    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)
    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)
    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)
    return train_ds, val_ds, test_ds

train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUT
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.A

num_filters = [32, 64, 64, 64, 64, 64]
kernel_sizes = [(3, 3)] * 6
conv_activation = 'relu'
dense_units = 64
dense_activation = 'relu'

def create_flexible_model(num_filters, kernel_sizes, conv_activation,
    resize_and_rescale = tf.keras.Sequential([
        layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_S
        layers.experimental.preprocessing.Rescaling(1.0/255)
    ])
    data_augmentation = tf.keras.Sequential([
        layers.experimental.preprocessing.RandomFlip("horizontal_and_v
        layers.experimental.preprocessing.RandomRotation(0.2)
    ])
    input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
    n_classes = 3
    model = models.Sequential()
    model.add(resize_and_rescale)
    model.add(data_augmentation)
    for filters, kernel_size in zip(num_filters, kernel_sizes):
        model.add(layers.Conv2D(filters, kernel_size=kernel_size, acti
        model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(dense_units, activation=dense_activation))
    model.add(layers.Dense(n_classes, activation='softmax'))
    return model

model = create_flexible_model(num_filters, kernel_sizes, conv_activati
model.build(input_shape=(None, IMAGE_SIZE, IMAGE_SIZE, CHANNELS))
model.summary()

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy'
history = model.fit(train_ds, validation_data=val_ds, epochs=50)

scores = model.evaluate(test_ds, verbose=0)
print(f"Test Loss: {scores[0]}, Test Accuracy: {scores[1]}")
```

Chapter 5

Results and Discussion

5.1 Model Performance

The CNN model was trained for 50 epochs with a batch size of 32. It achieved a test accuracy of 93.75% and a test loss of 0.083.

Metric	**Value**
Test Loss	0.083
Test Accuracy	93.75%

Table 5.1: Performance metrics of the CNN model on the test set.

5.2 Training and Validation Curves

The training and validation losses, along with the accuracy curves, were analyzed to evaluate the model's performance throughout the training process.

The loss curves show a steady decrease in both training and validation loss, indicating effective model convergence. The accuracy curves illustrate significant improvement, with the model achieving high accuracy on both training and validation sets.

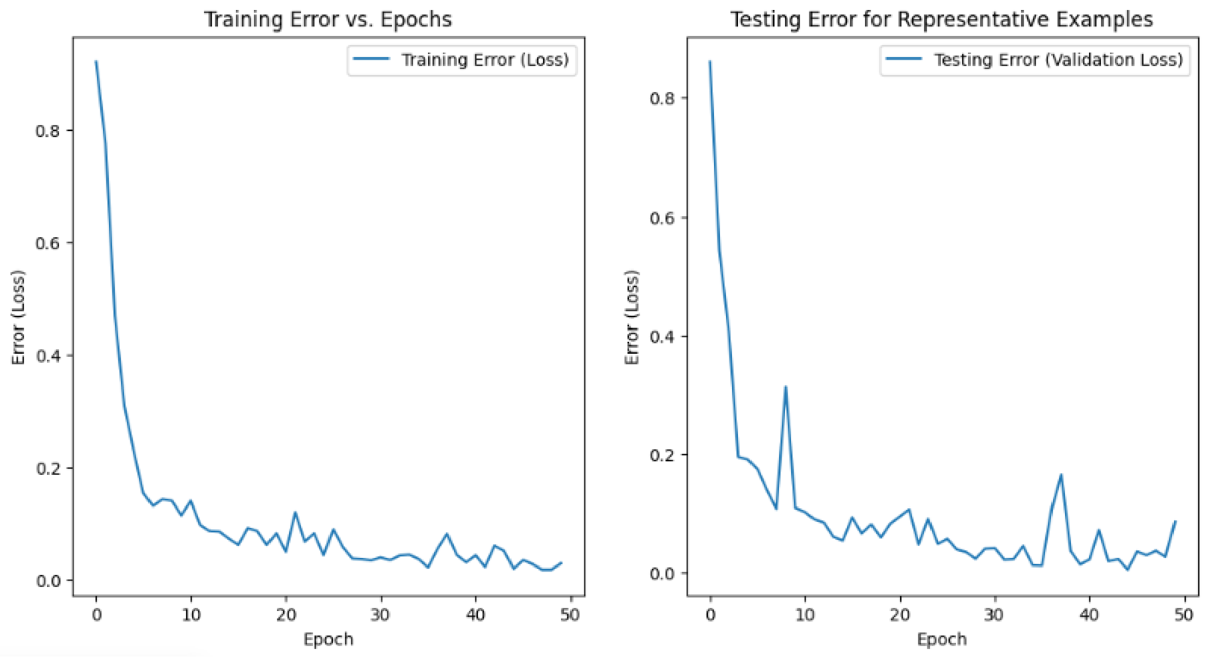

Figure 5.1: Plot of training and testing loss versus epochs.

5.3 Final Model Predictions

Figure 5.3 presents the predictions made by the final trained model on the test dataset. Each image includes the model's predicted label, the actual label, and the confidence score.

5.4 Discussion

The CNN model demonstrated strong performance with a high test accuracy of 93.75% and a low test loss of 0.083. The training and validation loss curves suggest effective learning with no significant overfitting. The accuracy curves indicate consistent improvement over the epochs.

Error analysis reveals that misclassifications primarily occurred between visually similar disease categories. This suggests potential areas for further enhancement in model architecture or data augmentation techniques to improve classification accuracy.

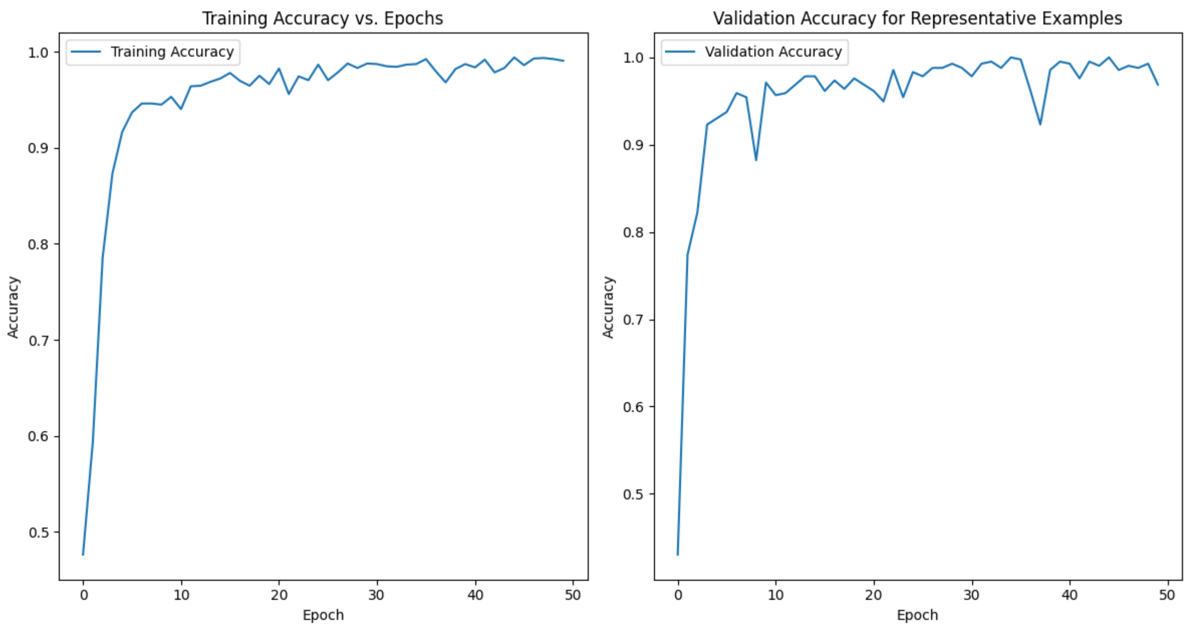

Figure 5.2: Plot of training and testing accuracy versus epochs.

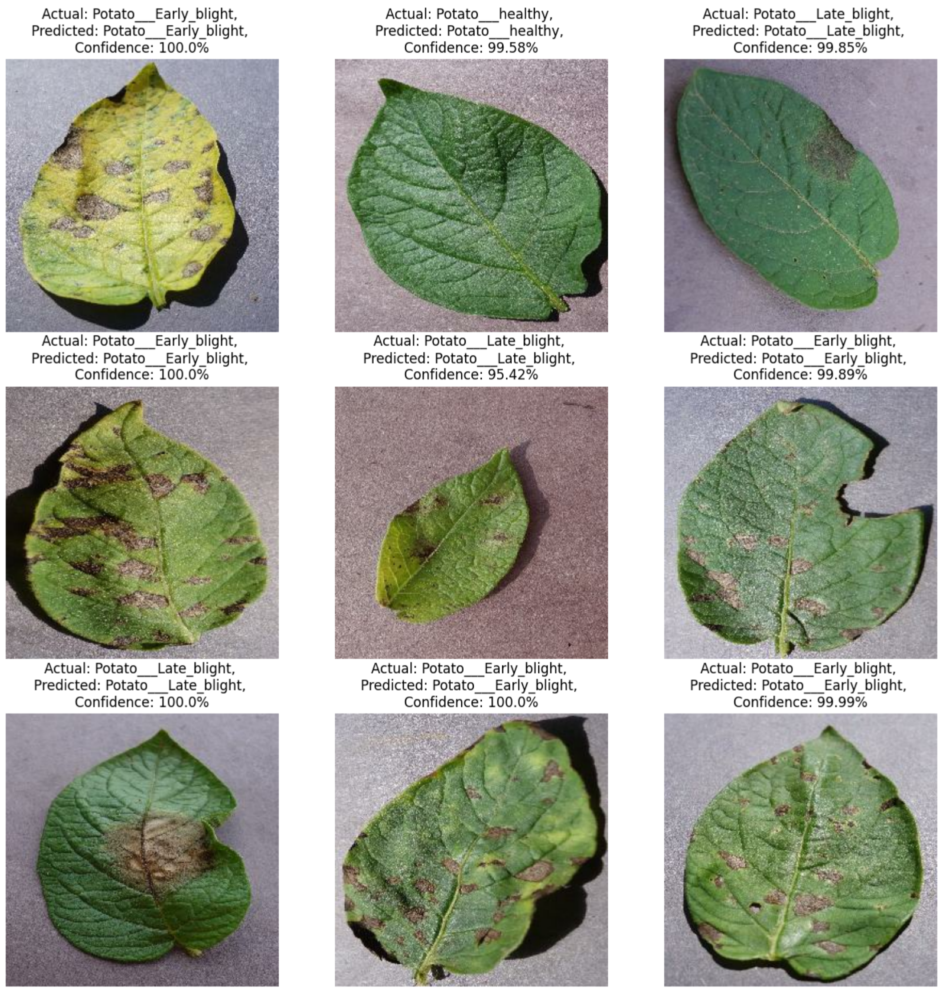
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence: 100.0%
Actual: Potato___healthy,
Predicted: Potato___healthy,
Confidence: 99.58%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight,
Confidence: 99.85%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence: 100.0%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight,
Confidence: 95.42%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence: 99.89%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight,
Confidence: 100.0%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence: 100.0%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence: 99.99%

Figure 5.3: Final model predictions on the test dataset. Each image shows the actual label, predicted label, and confidence score.

Chapter 6

Conclusion

In this project, TensorFlow and Keras were used to effectively build a Convolutional Neural Network (CNN) for the purpose of classifying potato plant diseases. Multiple convolutional, activation, and max-pooling layers were included in the architecture of the model, in addition to fully connected dense layers. To improve model generalization, preprocessing techniques like resizing, rescaling, and data augmentation were used.

The sparse categorical cross-entropy loss function and Adam optimizer were used to train the model across 50 epochs. A test loss of 0.083 and a test accuracy of 93.75% were found in the final examination, indicating good performance in the classification of diseases. The model's ability to learn and generalize from the data was validated using the training and validation metrics.

The effective deployment proved CNNs' potential for use in agricultural applications. Google Colab, TensorFlow, Keras, and Python offered a stable platform for model development and experimentation. In order to increase performance even further, future work may concentrate on additional data augmentation methods, optimizations, and the investigation of more sophisticated structures.