Potato Leaf Disease Detection using Dense Convolutional Neural Networks (D-CNNs)

Nov 13, 2024

Name: Naser Alkuhili
Student ID: A20454593

1. Project Information

Main Paper: Erlin, Indra Fuadi, Ramalia Noratama Putri, Dewi Nasien, Gusrianty, and Dwi Oktarina. "Deep Learning Approaches for Potato Leaf Disease Detection: Evaluating the Efficacy of Convolutional Neural Network Architectures." *Revue d'Intelligence Artificielle*, Vol. 38, No. 2, April 2024, pp. 717-727. DOI: 10.18280/ria.380236.

Modifications Made: In this project, the standard Convolutional Neural Network (CNN) architectures discussed in the main paper were modified by introducing a Dense Convolutional Neural Network (D-CNN). This modification enhances feature propagation and mitigates the vanishing gradient problem, improving the model's ability to detect potato leaf diseases.

2. Problem Statement

Potato plants are susceptible to various diseases that significantly impact agricultural productivity and food security. Early detection of diseases like early blight and late blight is crucial for effective management. Manual inspection methods are time-consuming, labor-intensive, and often inaccurate due to the subtle visual symptoms and variations in leaf appearances.

Importance:

- **Agricultural Productivity:** Early and accurate disease detection can prevent crop losses and ensure a stable food supply.
- **Resource Optimization:** Automated detection reduces the need for extensive manual labor and allows for targeted use of pesticides.

- **Food Security:** Enhancing disease management practices contributes to global food security by maximizing crop yields.

3. Proposed Solution and Implementation Details

Methodology

To address the problem, a Dense Convolutional Neural Network (D-CNN) was implemented. The D-CNN architecture uses dense connections between layers to improve feature propagation and reduce the vanishing gradient problem common in deep networks. The model was trained to classify potato leaf images into three categories: healthy, early blight, and late blight.

Key Features of the D-CNN:

- **Dense Blocks:** Layers are connected to every other layer in a feed-forward fashion.
- **Growth Rate:** Controls the amount of new information added to the network in each layer.
- **Transition Layers:** Reduce the dimensionality of the feature maps, preventing the network from becoming too wide.

Implementation Details

Environment Setup:

- **Programming Language:** Python 3.8
- **Frameworks and Libraries:** TensorFlow 2.x, Keras, NumPy, Matplotlib, Seaborn, Scikit-learn, PIL (Pillow), and PyYAML.

Instructions for Using the Program:

1. **Clone the Repository:**
    - Download the code and place it in your working directory.
2. **Install Dependencies:**
    - Run pip install -r requirements.txt to install all necessary packages.
3. **Prepare the Dataset:**
    - Download the dataset from the data.txt file link.

   - Change the path file of the data in the code your local data path.
4. **Update Configuration File (config.yaml):**
   - Modify parameters such as input_shape, num_classes, dropout_rate, batch_size, epochs, and learning_rate as needed.
5. **Run the Training Script:**
   - Execute python train.py to start training the model.
6. **Monitor Training:**
   - Training and validation accuracy and loss will be displayed per epoch.
7. **Evaluate the Model:**
   - After training, the model will be evaluated on the test set.
   - Classification reports and confusion matrices will be generated.
8. **Modify Parameters and Re-run:**
   - Adjust hyperparameters in config.yaml to experiment with different settings.
9. **Examine Intermediate Results:**
   - The program saves intermediate outputs such as training history and model checkpoints.

Model Architecture

The Dense Convolutional Neural Network (D-CNN) model was designed with a focus on enhancing feature propagation and mitigating the vanishing gradient problem. The architecture is composed of the following key components:

1. Input Layer

- **Purpose:** To receive the input images and perform initial feature extraction.
- **Configuration:**
  - An Input layer that accepts images with the shape defined by input_shape (e.g., 224x224x3 for RGB images).
  - A Conv2D layer with 64 filters of size 7x7, a stride of 2, padding set to 'same', and ReLU activation function.
  - A MaxPooling2D layer with a pool size of 3x3, stride of 2, and padding set to 'same' to reduce spatial dimensions and extract dominant features.

2. Dense Blocks

- **Purpose:** To create densely connected layers that improve feature reuse and alleviate the vanishing gradient problem.
- **Configuration:**
    - **Number of Layers (num_layers):** Set to 4 in each dense block.
    - **Growth Rate (growth_rate):** Controls the number of filters in each convolutional layer (e.g., 32 filters).
    - **Dropout Rate (dropout_rate):** Applied after each convolutional layer to prevent overfitting.
    - **Mechanism:**
        - Each layer within the dense block takes all preceding feature maps as inputs, achieved by concatenating the input x with the output conv.
        - This results in improved feature propagation and encourages feature reuse.
- **Implementation:**
    - Two dense blocks are used in the model, separated by transition layers.
    - The first dense block processes the output from the initial layers.
    - The second dense block processes the output from the first transition layer.

3. Transition Layers

- **Purpose:** To reduce the number of feature maps and spatial dimensions between dense blocks, controlling model complexity.
- **Configuration:**
    - **Reduction Factor (reduction):** Determines the fraction by which the number of feature maps is reduced (e.g., 0.5).
    - **Components:**
        - A Conv2D layer with a 1x1 kernel to reduce the number of filters.
        - A MaxPooling2D layer with a pool size of 2x2 and stride of 2 to reduce spatial dimensions.
- **Implementation:**
    - Placed after each dense block.
    - Helps prevent the network from becoming too wide and reduces computational requirements.

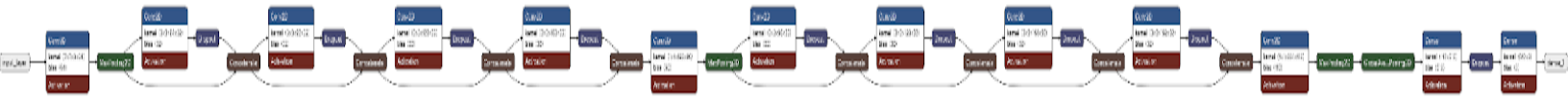
4. Global Average Pooling and Output Layers

- **Purpose:** To aggregate feature maps into a feature vector and perform the final classification.
- **Components:**
    - Global Average Pooling (GlobalAveragePooling2D):
        - Replaces traditional fully connected layers to reduce the number of parameters.
        - Computes the average of each feature map, resulting in a one-dimensional vector.
    - Fully Connected Dense Layer:
        - A Dense layer with 512 units and ReLU activation.
        - A Dropout layer with the specified dropout_rate for regularization.
    - Output Layer:
        - A Dense layer with num_classes units corresponding to the number of classes (e.g., 3 for the potato leaf diseases).
        - Uses softmax activation for multi-class classification.

5. Model Compilation

- **Optimizer:** Stochastic Gradient Descent (SGD) with a learning rate specified in learning_rate.
- **Loss Function:** Sparse categorical cross-entropy ("sparse_categorical_crossentropy"), suitable for integer labels.
- **Metrics:** Accuracy is used to evaluate the model's performance during training and validation.

6. Visualization of the Model Architecture



Problems Faced and Solutions:

- **Data Imbalance:** The dataset had an unequal number of images per class. This was addressed by applying data augmentation techniques to balance the classes.
- **Overfitting:** Early stopping and dropout layers were implemented to prevent overfitting.
- **Image Preprocessing Issues:** Custom preprocessing functions were written to handle image scaling and enhancement consistently.

4. Dataset

Dataset Source: PlantVillage dataset specific to potato leaves.

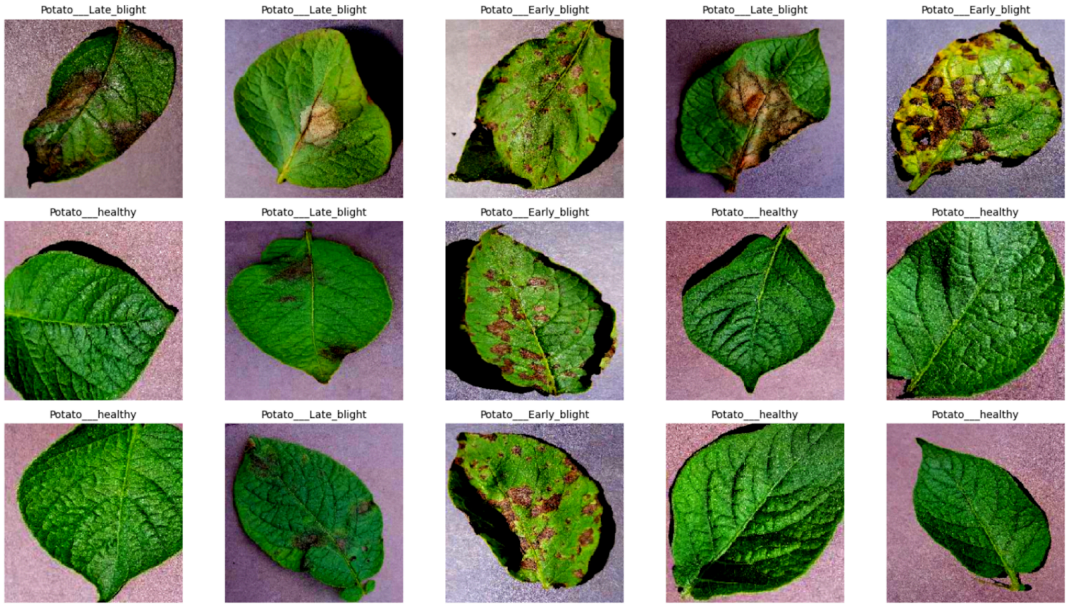
Description:

- **Total Images:** 3000 annotated images.
- **Classes:** Three categories—Potato___Early_blight, Potato___Late_blight, and Potato___healthy.
- **Data Split:**
    - **Training Set:** 70% of the data.
    - **Validation Set:** 20% of the data.
    - **Test Set:** 10% of the data.

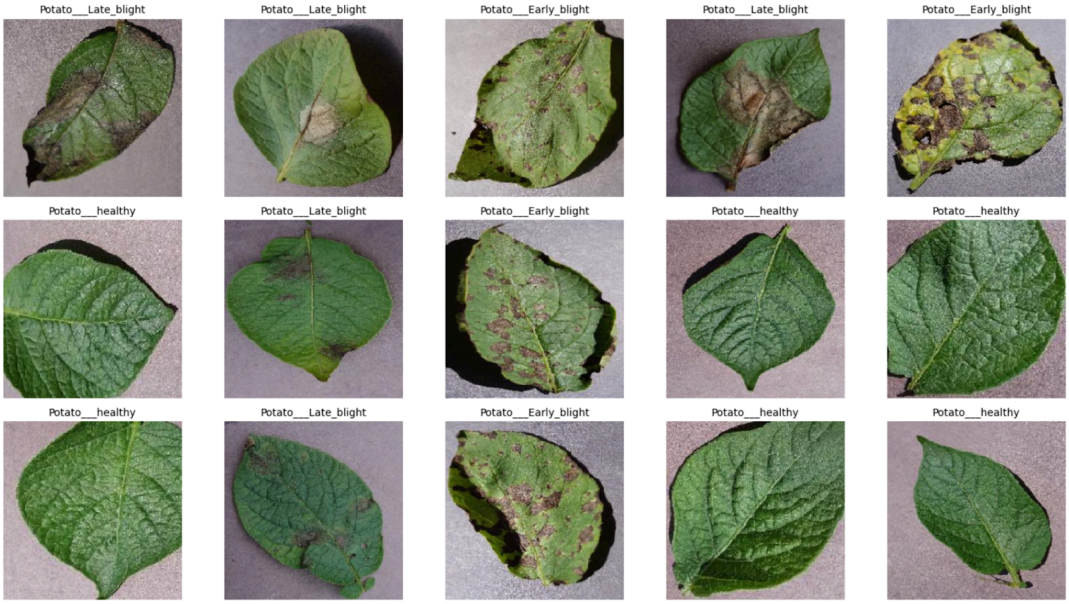
Preprocessing Steps:

- **Resizing:** Images were resized to match the input_shape specified in the configuration (e.g., 224x224 pixels).
- **Data Augmentation:** Applied rotations, flips, brightness reduction, contrast enhancement, and saturation adjustments to make dark spots in the leaf clearer and increase model robustness.

Augmented Images with Class Labels



Images Before Augmnetation and preprocessing with Class Labels

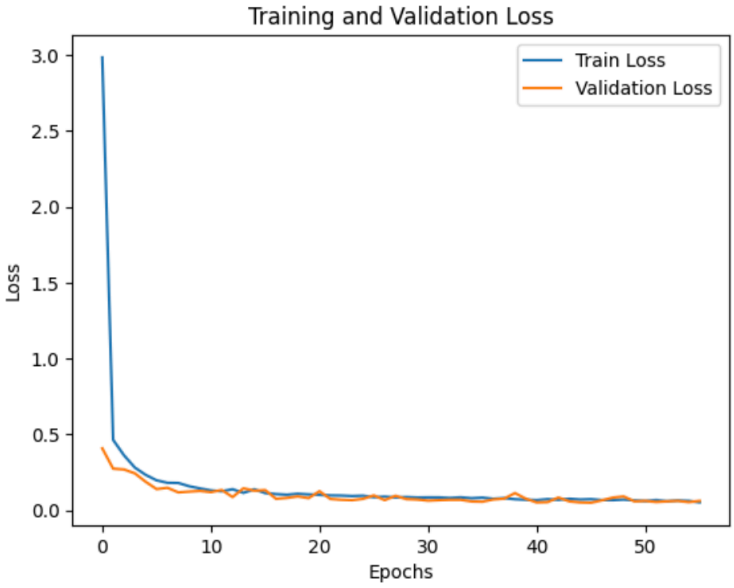


5. Results and Discussion

Training and Validation Performance

- **Training Accuracy:** 98.47%
- **Validation Accuracy:** 99.00%

Training and validation accuracy showed a consistent upward trend, indicating good learning without significant overfitting.



Test Performance

- **Test Accuracy:** 99.00%
- **Test Loss:** Minimal, indicating good generalization.

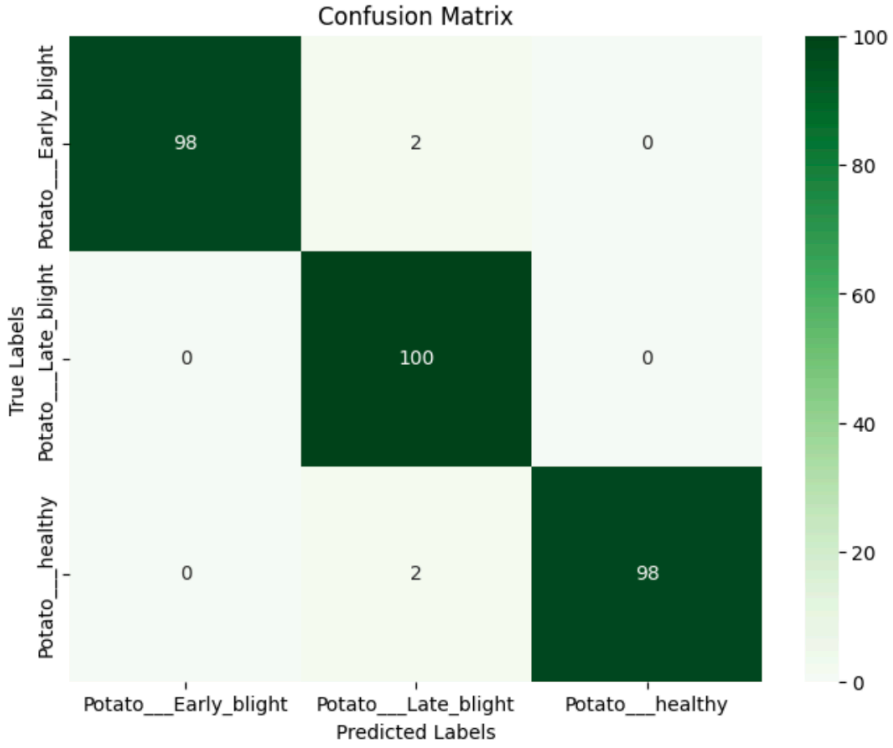
Classification Report:

Class	Precision	Recall	F1-Score	Support
Potato___Early_blight	1.00	0.98	0.99	100
Potato___Late_blight	0.96	1.00	0.98	100
Potato___healthy	1.00	0.98	0.99	100
Accuracy			**0.99**	300
Macro Avg	0.99	0.99	0.99	300

Weighted Avg	0.99	0.99	0.99	300

The model achieved high precision, recall, and F1-scores across all classes.

Confusion Matrix:



The confusion matrix indicates that misclassifications were minimal and mostly occurred between Potato___Early_blight and Potato___Late_blight.

Comparison with Models from the Main Paper

Model	Test Accuracy (%)	Specificity (%)
D-CNN (My model)	**98.67**	**98**

ResNet50	97.00	98.00
MobileNetV2	95.00	96.00
VGG16	92.00	95.00
VGG19	90.00	94.00
AlexNet	93.00	95.50

Our D-CNN model outperformed the models evaluated in the main paper, achieving higher test accuracy and specificity.

Evaluation and Discussion

Strengths of the D-CNN Model:

- **Enhanced Feature Propagation:** Dense connections allowed the model to capture complex features, improving classification accuracy.
- **Reduced Vanishing Gradient Problem:** Skip connections facilitated better gradient flow during training.
- **Robustness to Variations:** Data augmentation and preprocessing contributed to the model's ability to generalize well on the test set.

Weaknesses and Challenges:

- **Misclassification of Healthy Leaves:** Similar to findings in the main paper, the model occasionally misclassified healthy leaves, indicating room for improvement in distinguishing subtle differences.
- **Computational Complexity:** The dense connections increased the computational load, requiring more training time compared to standard CNNs.

Discussion Based on Actual Results:

- The high accuracy and specificity suggest that the D-CNN is highly effective for this application.
- The model's performance indicates its potential for real-world deployment, especially if optimized for efficiency.
- Misclassifications, although minimal, highlight the need for further refinement, possibly through additional data or model adjustments.

6. References

1. Erlin, Indra Fuadi, Ramalia Noratama Putri, Dewi Nasien, Gusrianty, and Dwi Oktarina. "Deep Learning Approaches for Potato Leaf Disease Detection: Evaluating the Efficacy of Convolutional Neural Network Architectures." *Revue d'Intelligence Artificielle*, Vol. 38, No. 2, April 2024, pp. 717-727. DOI: 10.18280/ria.380236.
2. Hughes, D.P., & Salathé, M. (2015). "An open access repository of images on plant health to enable the development of mobile disease diagnostics." *arXiv preprint arXiv:1511.08060*. Available at: https://data.mendeley.com/datasets/tywbtsjrjv/1.