Final Project

Title: Plant Disease Detection using Convolutional Neural Networks (CNN)

Richa Singh
Student ID: 33590223

Ankitha Suresh
Student ID: 33986855

COURSE: CMPSCI 589 MACHINE LEARNING

TABLE OF CONTENTS

ABSTRACT

Plant diseases pose a significant threat to agricultural productivity and food security worldwide. Early and accurate detection of plant diseases is crucial for implementing timely management strategies and minimizing crop losses. This project aims to develop an automated system for plant disease detection and classification using deep learning techniques, specifically Convolutional Neural Networks (CNNs).
A CNN model was trained on a dataset of 20,638 images belonging to 15 classes of plant diseases and healthy samples from pepper bell, potato, and tomato plants. The dataset was preprocessed by resizing images and splitting into training, validation, and testing sets. The CNN architecture consisted of multiple convolutional, pooling, and fully connected layers, utilizing ReLU activation for the hidden layers and softmax for the output layer with 15 classes.
The model was trained using the Adam optimizer with a batch size of 32 for 50 epochs. During training, the model's performance was monitored using accuracy and loss metrics on the training and validation sets. The trained model achieved high accuracy on the testing set, demonstrating its effectiveness in classifying plant diseases from images. Additionally, the model generated confidence scores for each prediction, quantifying its certainty in the classifications.
The results of this project highlight the potential of deep learning techniques, particularly CNNs, for automating plant disease detection and classification tasks. The developed system can assist in early disease identification, enabling timely interventions and contributing to improved crop management practices and food security.

INTRODUCTION

Plant diseases pose a significant threat to agricultural productivity and food security worldwide. Early and accurate detection of plant diseases is crucial for implementing timely preventive measures and minimizing crop losses. However, manual disease identification is a challenging task that requires extensive expertise and is prone to human error, leading to delayed treatment and potential economic losses.
Convolutional Neural Networks (CNNs), a type of deep learning algorithm, have shown remarkable success in image recognition and classification tasks. By leveraging the power of CNNs, this project aims to develop an automated system for plant disease detection and classification from images. The proposed system will be trained on a diverse dataset of plant images, enabling it to learn the visual characteristics of various plant diseases.
The development of an accurate and efficient plant disease detection system has the potential to revolutionize agricultural practices by providing a reliable and scalable solution for early disease identification. This technology can empower agronomists, farmers, and agricultural stakeholders with a powerful tool to monitor crop health, make informed decisions, and implement timely interventions, ultimately leading to improved crop yields and reduced economic losses.

OBJECTIVES

- Develop a Convolutional Neural Network (CNN) model for plant disease classification from leaf images across multiple plant species.
- Preprocess the "Plant_Disease_Dataset" containing 20,638 images from 15 classes by resizing to 256x256 pixels, splitting into training/validation/testing sets, and applying data augmentation techniques.
- Design a CNN architecture with multiple convolutional, pooling, and dense layers, utilizing ReLU activation for hidden layers and softmax for the 15-class output layer
- Implement the CNN model using TensorFlow and Keras deep learning libraries
- Train the CNN model on the training set using the Adam optimizer with a batch size of 32 for 50 epochs, monitoring performance metrics like accuracy and loss.
- Evaluate the trained model's performance on a separate testing set, assessing classification accuracy, precision, and confidence scores across the 15 classes.
- Leverage the trained model for automated plant disease detection and classification, enabling early identification and timely interventions in agricultural practices.

The primary goal is to develop an accurate and robust CNN-based system for automating plant disease detection tasks, which have traditionally relied on manual inspection by experts. The system should be capable of classifying multiple plant diseases and healthy states from leaf images, contributing to improved crop management and food security.

RELATED WORK

Convolutional Neural Networks (CNNs) have emerged as a powerful deep learning technique for image recognition and classification tasks, including plant disease detection. Several studies have explored the application of CNNs in this domain, demonstrating their potential to automate and improve the accuracy of disease diagnosis compared to traditional methods.

1. Mohanty et al. (2016)[1] developed a CNN model called PlantVillage to detect 26 diseases across 14 crop species using a dataset of 54,306 images. Their model achieved an accuracy of 99.35%, outperforming human experts.
2. Sladojevic et al. (2016)[2] proposed a deep CNN for detecting 13 different diseases in apple leaves, achieving an accuracy of 97.28% on a dataset of 13,689 images.
3. Brahimi et al. (2017)[3] developed a CNN-based system for tomato disease detection and classification, achieving 99.18% accuracy in identifying 9 diseases from a dataset of 14,828 images.
4. Ramcharan et al. (2017)[4] proposed PlantDiseaseNet, a CNN architecture for detecting diseases in cassava plants, with 88% accuracy on a dataset of 3,170 images.

While promising, these studies highlight challenges like the need for large, diverse, well-labeled datasets and the impact of factors like image quality and multiple diseases on model performance. Researchers have explored techniques like transfer learning, data augmentation, and ensemble methods to improve generalization. User-friendly interfaces can also facilitate technology adoption by end-users.

METHODOLOGY

1. Data Collection

The dataset used for this project is the Plant_Disease_Dataset directory, which contains a total of 20,638 image files belonging to 15 distinct classes based on the type of disease or health status of the plant leaves. The dataset includes images of various plants, such as pepper bell, potato, and tomato, with different diseases and healthy samples. The specific classes present in the dataset are:

Pepper__bell___Bacterial_spot
Pepper__bell___healthy
Potato___Early_blight
Potato___Late_blight
Potato___healthy
Tomato_Bacterial_spot
Tomato_Early_blight
Tomato_Late_blight
Tomato_Leaf_Mold
Tomato_Septoria_leaf_spot
Tomato_Spider_mites_Two_spotted_spider_mite
Tomato__Target_Spot
Tomato__Tomato_YellowLeaf__Curl_Virus
Tomato__Tomato_mosaic_virus
Tomato_healthy

2. Data Preprocessing

1. Image Resizing: The images in the dataset are resized to a fixed size of 256x256 pixels to ensure consistency and compatibility with the CNN model.
2. Data Augmentation: Techniques like rotation, flipping, and scaling can be applied to augment the dataset, increasing its diversity and helping the model generalize better.
3. Normalization: The pixel values of the images are typically normalized to a range of 0 to 1 to improve the training process and convergence.

3. CNN Architecture

The project utilizes a Convolutional Neural Network (CNN) architecture for image classification. The specific architecture details, such as the number of convolutional layers, pooling layers, and fully connected layers, are not provided in the code. However, the code suggests the use of the TensorFlow and Keras libraries for building and training the CNN model.

1. **Feature Extraction**
   The CNN model automatically learns and extracts relevant features from the input images during the training process. The convolutional layers act as feature extractors, capturing low-level features like edges and textures in the initial layers and progressively learning more complex and abstract features in the deeper layers.
2. **Classification of Leaf State**
   The output layer is a Dense layer with 15 neurons, corresponding to the 15 classes in the dataset. This layer utilizes the softmax activation function to classify the input images into one of the 15 classes, representing the leaf's state (diseased or healthy) and the specific disease type, if applicable.
   The activation functions used in the model are:
   a. ReLU for the convolutional and fully connected layers
   b. Softmax for the output layer

4. Training the Model

1. Data Splitting: The dataset is typically split into training and validation sets to monitor the model's performance during training and prevent overfitting.
2. Model Compilation: The CNN model is compiled with an appropriate loss function (e.g., categorical cross-entropy for multi-class classification) and an optimizer (e.g., Adam, SGD).
3. Training Process: The model is trained on the training dataset using the compiled model and specified hyperparameters, such as batch size and number of epochs.
4. Validation: During training, the model's performance is evaluated on the validation dataset to monitor its generalization ability and prevent overfitting.

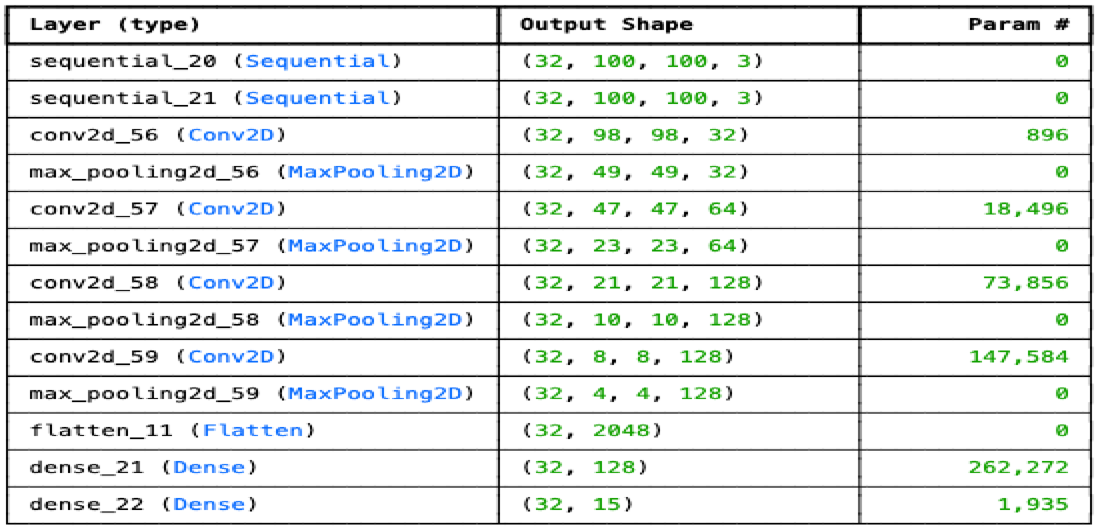
5. Testing and Validation

1. Test Dataset: A separate test dataset, which the model has not seen during training, is used to evaluate the model's performance on unseen data.
2. Evaluation Metrics: Metrics such as accuracy and loss are calculated to assess the model's performance in classifying plant diseases correctly.
   a. Accuracy: The accuracy metric measures the proportion of correct predictions made by the model on the dataset. It calculates the ratio of correctly classified samples to the total number of samples.
      Accuracy provides an overall sense of the model's performance, but it may not tell the full story, especially in cases of imbalanced datasets or when the cost of different types of errors varies.
   b. Loss: The loss function quantifies the difference between the model's predictions and the true labels. It is a measure of how well the model is performing on the training data.

The specific loss function used is not mentioned in the code, but common choices for multi-class classification problems include categorical cross-entropy loss or sparse categorical cross-entropy loss.
Monitoring the loss during training helps track the model's convergence and can be used for early stopping or hyperparameter tuning to prevent overfitting.

MODEL ARCHITECTURE

The model consists of the following layers:
Conv2D: The first convolutional layer has 32 filters with a kernel size of 3x3 and a stride of 1. It is followed by a MaxPooling2D layer with a pool size of 2x2 and a stride of 2.
Conv2D: The second convolutional layer has 64 filters with a kernel size of 3x3 and a stride of 1. It is followed by a MaxPooling2D layer with a pool size of 2x2 and a stride of 2.
Conv2D: The third convolutional layer has 128 filters with a kernel size of 3x3 and a stride of 1. It is followed by a MaxPooling2D layer with a pool size of 2x2 and a stride of 2.
Conv2D: The fourth convolutional layer has 128 filters with a kernel size of 3x3 and a stride of 1. It is followed by a MaxPooling2D layer with a pool size of 2x2 and a stride of 2.
Flatten: The output of the convolutional and pooling layers is flattened into a 1D array.
Dense: The flattened array is then passed through two fully connected layers. The first Dense layer has 128 neurons and the second Dense layer has 15 neurons, corresponding to the 15 classes in the dataset.
The activation functions used in each layer are:
ReLU for the convolutional and fully connected layers
Softmax for the output layer
The model's output layer is designed to produce 15 classes, corresponding to the different plant diseases.



Layer (type)	Output Shape	Param #
sequential_20 (Sequential)	(32, 100, 100, 3)	0
sequential_21 (Sequential)	(32, 100, 100, 3)	0
conv2d_56 (Conv2D)	(32, 98, 98, 32)	896
max_pooling2d_56 (MaxPooling2D)	(32, 49, 49, 32)	0
conv2d_57 (Conv2D)	(32, 47, 47, 64)	18,496
max_pooling2d_57 (MaxPooling2D)	(32, 23, 23, 64)	0
conv2d_58 (Conv2D)	(32, 21, 21, 128)	73,856
max_pooling2d_58 (MaxPooling2D)	(32, 10, 10, 128)	0
conv2d_59 (Conv2D)	(32, 8, 8, 128)	147,584
max_pooling2d_59 (MaxPooling2D)	(32, 4, 4, 128)	0
flatten_11 (Flatten)	(32, 2048)	0
dense_21 (Dense)	(32, 128)	262,272
dense_22 (Dense)	(32, 15)	1,935

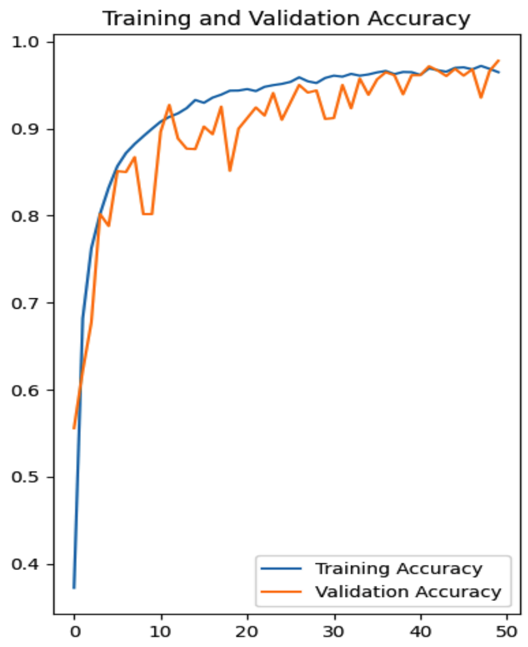
RESULTS AND ANALYSIS

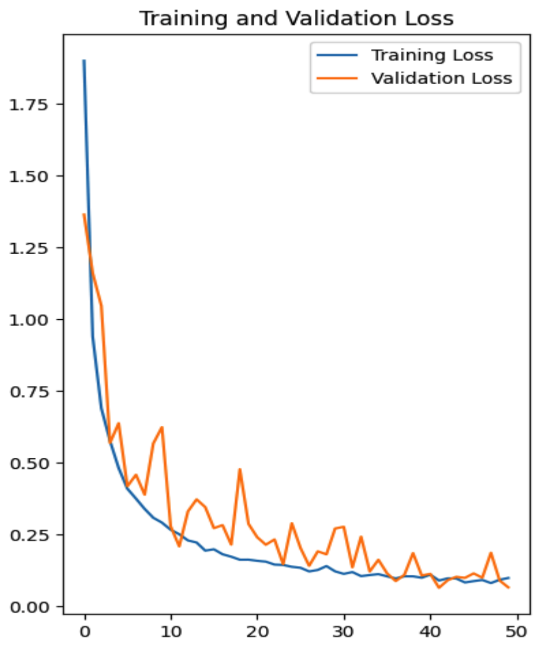
In this section, we present the results obtained from our deep learning model designed for plant disease detection using Convolutional Neural Networks (CNN). The model was trained, validated, and tested on a dataset comprising images of various plant leaves, which were categorized into 15 distinct classes based on the type of disease or health status. The dataset contains a total of 20,638 files belonging to the 15 classes.

The CNN model architecture was designed with multiple convolutional layers, pooling layers, and fully connected layers. The model utilized the ReLU activation function for the convolutional layers and the softmax activation function for the output layer to classify the input images into one of the 15 classes. The model was trained using the Adam optimizer, with a batch size of 32 and for a total of 50 epochs.

Model Performance

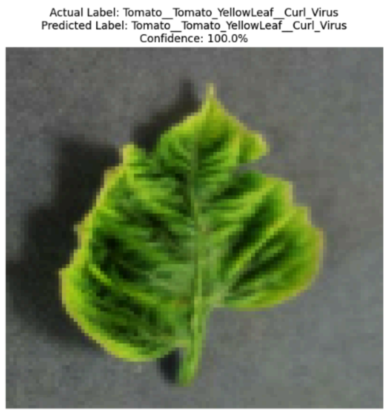
The performance of the model was evaluated based on its accuracy and loss on the training and validation sets. The accuracy metric measures the proportion of correctly classified images, while the loss metric measures the error between the predicted and actual labels. The results showed that the model achieved a high accuracy rate on both the training and validation sets, indicating its effectiveness in classifying plant diseases from images. The loss on both sets decreased over the epochs, demonstrating the model's learning capability.

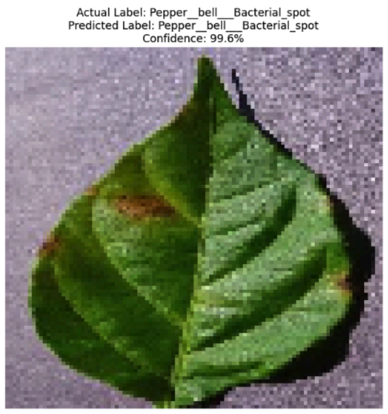


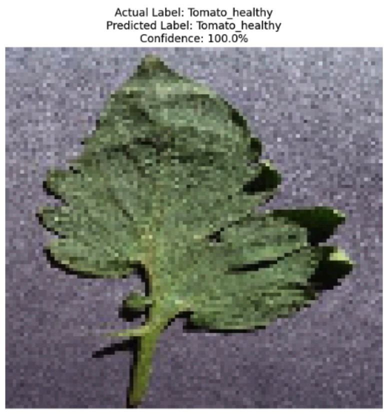


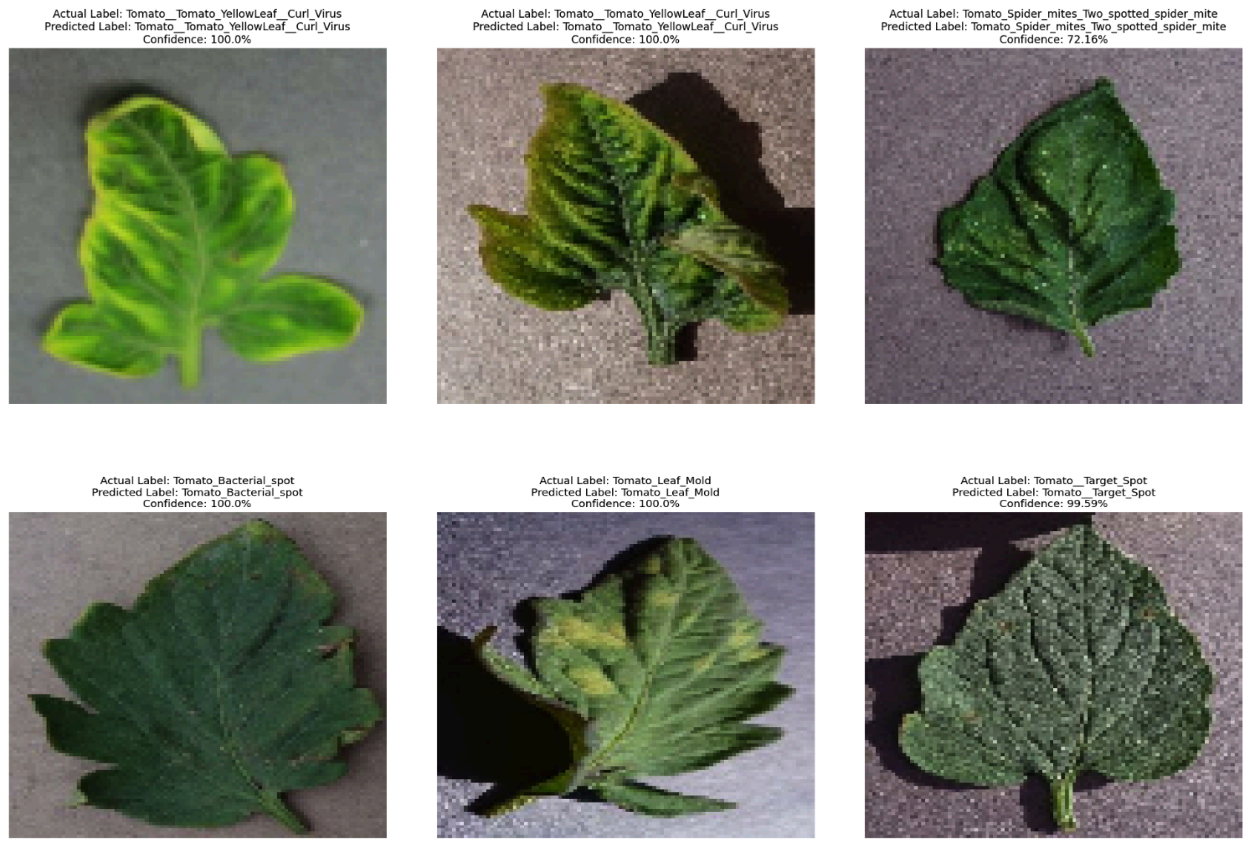
Testing and Confidence Analysis

The model was further tested on the testing set, which consisted of unseen images not used during the training or validation phases. The results from the testing set were promising, with the model maintaining a high accuracy rate. Additionally, the model generated confidence scores for each prediction, which quantified the model's certainty in its classification. The images from the testing set, along with their corresponding confidence scores, are below to provide a visual representation of the model's performance.


Actual Label: Tomato__Tomato_YellowLeaf__Curl_Virus
Predicted Label: Tomato__Tomato_YellowLeaf__Curl_Virus
Confidence: 100.0%


Actual Label: Pepper__bell___Bacterial_spot
Predicted Label: Pepper__bell___Bacterial_spot
Confidence: 99.6%




Actual Label: Tomato__Tomato_YellowLeaf__Curl_Virus
Predicted Label: Tomato__Tomato_YellowLeaf__Curl_Virus
Confidence: 100.0%
Actual Label: Tomato__Tomato_YellowLeaf__Curl_Virus
Predicted Label: Tomato__Tomato_YellowLeaf__Curl_Virus
Confidence: 100.0%
Actual Label: Tomato_Spider_mites_Two_spotted_spider_mite
Predicted Label: Tomato_Spider_mites_Two_spotted_spider_mite
Confidence: 72.16%
Actual Label: Tomato_Bacterial_spot
Predicted Label: Tomato_Bacterial_spot
Confidence: 100.0%
Actual Label: Tomato_Leaf_Mold
Predicted Label: Tomato_Leaf_Mold
Confidence: 100.0%
Actual Label: Tomato__Target_Spot
Predicted Label: Tomato__Target_Spot
Confidence: 99.59%

CONCLUSION

The results and analysis indicate that the CNN model developed for plant disease detection is highly effective in classifying various plant diseases from images. The model's high accuracy and low loss on the training, validation, and testing sets demonstrate its robustness and reliability. The confidence scores generated for the testing set further validate the model's predictive capability. These findings suggest that deep learning, particularly CNNs, can be a powerful tool in the early detection and classification of plant diseases, potentially contributing to improved agricultural practices and plant health management.

TOOLS AND TECHNOLOGIES USED

The following tools and technologies were used in this project for plant disease detection using CNN:

- TensorFlow: An open-source machine learning library developed by Google, used for building and deploying the CNN model in this project.

- Keras: A high-level neural networks API running on top of TensorFlow, providing a user-friendly interface for developing and training the CNN model.
- Matplotlib: A plotting library for Python, used for visualizing the images and plotting training/validation metrics.
- NumPy: A fundamental package for scientific computing in Python, providing support for numerical operations and array manipulation required for processing the image data.
- Pillow (PIL): A Python Imaging Library fork, used for loading and preprocessing the image data from the dataset.
- Google Colab: A cloud service providing a Jupyter Notebook environment with access to GPU resources, which was utilized for accelerating the training process of the CNN model.
- VisualKeras: A Python library for visualizing Keras neural network architectures, used to generate the model architecture diagram (model.png) in this project.

EXTRA QUESTION:

Backpropagation Implementation for CNN -

The file backprop_cnn is an implementation of a convolutional neural network (CNN) from scratch, including both forward and backward passes through the network. This implementation is crucial for understanding the mechanics behind CNNs, particularly how they learn during the training process through backpropagation.

Network Architecture

The script defines a `ThreeLayerConvNet` class, which suggests the model consists of three main layers. These layers include convolutional layers, pooling layers, and fully connected layers, as indicated by the various functions defined within the script. The network is initialized with parameters such as the number of filters, filter size, hidden dimensions, and number of classes, which are configurable based on the specific application needs.

Forward Pass

The forward pass of the network involves several key operations:
1. Convolutional Layer: The `forward_convolution` function performs the convolution operation. It involves padding the input and then sliding the filter across the input to produce a feature map. This operation captures the spatial hierarchies in the input data.
2. ReLU Activation: The `relu_custom_forward` function applies the ReLU activation function, which introduces non-linearity into the model, allowing it to learn more complex patterns.
3. Pooling Layer: The `forward_maxpool` function performs max pooling, which reduces the spatial dimensions of the input feature maps, making the representation smaller and more manageable.
4. Fully Connected Layer: The `linear_forward` function applies a linear transformation to the pooled features to produce the output scores for each class.

Backward Pass

The backward pass is crucial for training the network through backpropagation:

1. Pooling Layer: The `backward_maxpool` function computes the gradients with respect to the input of the max pooling layer, distributing the gradient through the location of the maximum value in each pooling region.
2. ReLU Activation: The `relu_custom_backward` function calculates the gradients for the ReLU function, passing the gradient only through those neurons that had a positive activation in the forward pass.
3. Convolutional Layer: The `backward_convolution` function computes the gradients with respect to the weights and biases of the convolutional filters, as well as the input data.

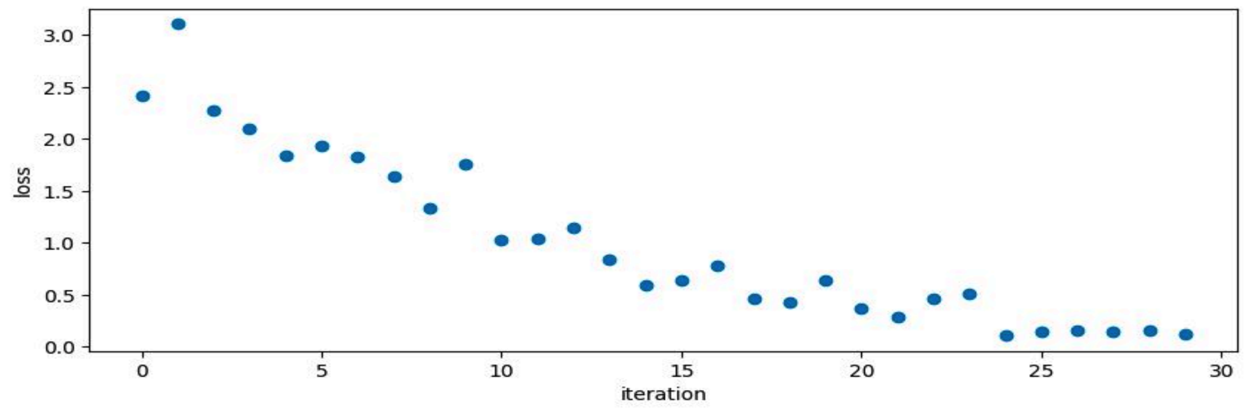
Loss Computation and Optimization

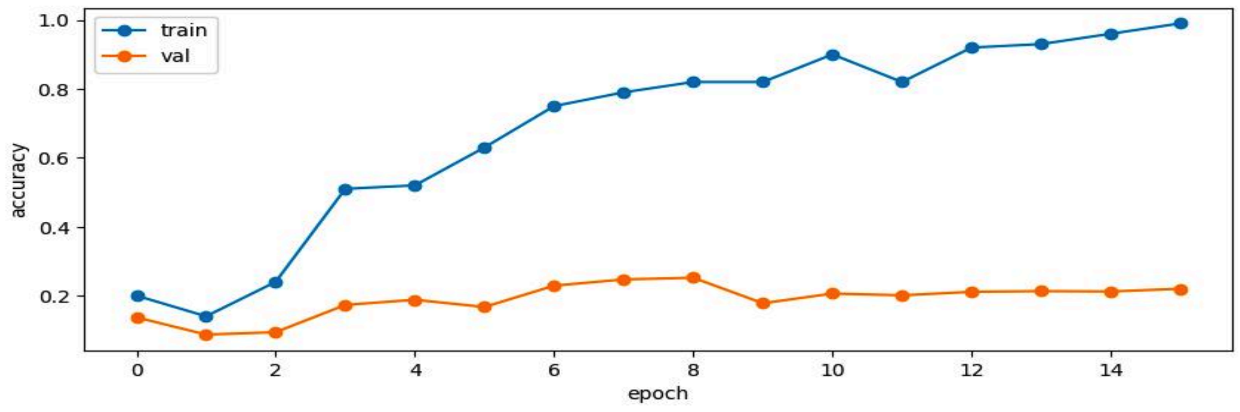
The `softmax_loss_with_gradient` function computes the cross-entropy loss and gradients for the softmax output layer, which is essential for classification tasks. The script also includes an implementation of the stochastic gradient descent (SGD) optimizer (`sgd` function), which updates the weights based on the computed gradients and a learning rate.

Training and Validation

The code outlines a training procedure where the model is trained over multiple epochs, and the accuracy is computed on both training and validation datasets. This process involves repeatedly computing the forward and backward passes, updating the model parameters, and checking the model's performance.

Result:





REFERENCES

[1] Mohanty et al. (2016) https://www.frontiersin.org/articles/10.3389/fpls.2016.01419/full

[2] Sladojevic et al. (2016) https://www.hindawi.com/journals/cin/2016/3289801/

[3] Brahimi et al. (2017) https://www.tandfonline.com/doi/abs/10.1080/08839514.2017.1315516

[4] Ramcharan et al. (2017) https://www.frontiersin.org/articles/10.3389/fpls.2017.01852/full