Smartphone-Assisted Crop Disease Diagnosis: Leveraging Deep Learning for Rapid Identification

INFO8010-1 : DEEP LEARNING

Professor

Pr. Gilles Louppe

Author

Wilfried Mvomo Eto - s226625

Academic year 2023 - 2024

Introduction

Human society needs to increase its food production by about 70% by 2050 to feed a population estimated at over 9 billion people. Currently, infectious diseases reduce potential yields by an average of 40%, with many farmers in developing countries experiencing yield losses of up to 100% [5]. Crop diseases pose a major threat to food security, but their rapid identification remains a challenge in many parts of the world due to a lack of necessary infrastructure.

Automated detection of plant diseases is essential as it reduces the laborious task of monitoring large farms and enables the early detection of diseases, thereby minimizing any further crop degradation. Besides the decline in plant health, a country's economy is heavily impacted by this scenario due to decreased production. The current approach of disease identification by an expert is slow and suboptimal for large farms. Models have been proposed, including a set of pre-trained AlexNet, DenseNet121, EfficientNetB7, and EfficientNet NoisyStudent, aimed at classifying apple tree leaves into one of the following categories: healthy, apple scab, apple rust, and multiple diseases, using their images.

In this project, we will construct our own Convolutional Neural Network (CNN) mainly based on the techniques outlined in the book "Deep Learning with PyTorch" [7]. Our CNN aims to achieve over 92% accuracy on both validation and test sets for identifying crop diseases from images captured by smartphones. We will develop a CNN model tailored to our specific needs. This model is designed to classify 38 class labels. Each label corresponds to a crop-disease pair, and our goal is to predict this pair from the image of the plant leaf. Various image augmentation techniques will be included in this research to increase the size of the dataset and, consequently, the model's accuracy.

To enhance the accuracy of our model, various image augmentation techniques will be employed to increase the size and diversity of the dataset. The aim is to provide farmers with a powerful and accessible tool for diagnosing crop diseases quickly and accurately, thereby contributing to improving global food security and agricultural yields.

By integrating advanced deep learning technologies with the convenience of smartphones, our project aims to achieve high accuracy and offer a practical and efficient solution for early detection of crop diseases.

1. Data Preparation:

We are examining a dataset consisting of 54,306 images depicting plant leaves, which are classified into 38 distinct categories (figure 1). Each category represents a specific crop-disease combination, and our objective is to accurately predict these combinations based on the leaf images. The original images have an average size of 256.0x256.0 pixels across all three versions of the dataset: color, segmented, and grayscale. These images have been sourced from the dataset available at the following repository: https://github.com/digitalepidemiologylab/plantvillage_deeplearning_paper_dataset.

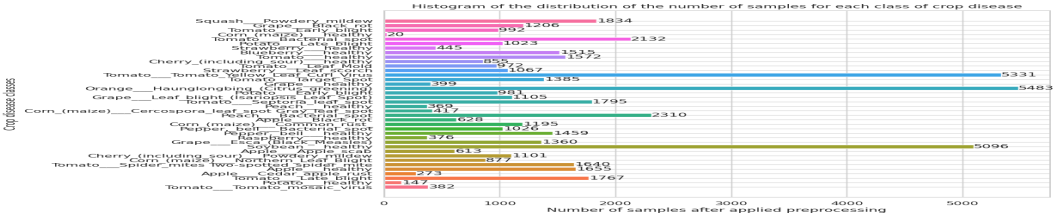

Figure 1: Histogram of the distribution of the number of samples for each class of crop disease

i. Expansion of our dataset to improve model performance:

To enhance our model's generalization ability and its robustness to image variations, we expanded our dataset using various image augmentation techniques. These techniques aim to create artificial variations in the training data to expose the model to a greater diversity of possible scenarios during inference. We applied several transformations to the images, including random vertical and horizontal flips, as well as random adjustments of brightness, contrast, saturation, and hue. Additionally, we resized all images to a uniform size of 120x120 pixels to ensure consistency in the model's input data. Besides we performed data preprocessing steps, including removing images with anomalies or inconsistencies, to ensure the quality and integrity of our dataset. Furthermore, to explore the effect of color on the model's performance, we also created a grayscale version of our dataset by converting color images to grayscale images. This allows us to compare the model's performance on color and grayscale data and evaluate the impact of color on its prediction capability. Finally, we also worked with a segmented version of our dataset. A segmented image is an image in which only the relevant parts of the image are retained, while the rest is removed or masked. This approach allows us to focus the model's attention on the specific part of the image containing the object of interest, by eliminating noise and distractions that could affect the model's performance.

By combining these different data augmentation approaches, we aim to improve the model's ability to generalize to new data and enhance its accuracy in detecting crop diseases (figure 1). When facing imbalanced classes in the histogram, it indicates that certain crop diseases are represented more frequently in the dataset compared to others. This imbalance can potentially lead to biases in the model's training, where the model might become overly specialized in predicting the more prevalent classes while neglecting the less represented ones. To address this issue, techniques such as data augmentation, resampling methods, or adjusting class weights during training can be employed to ensure that the model learns from all classes equally.

ii. Data Normalization

Data normalization is of crucial importance. This step involves calculating the means and standard deviations of pixel values in images to standardize them. Reducing the variation in pixel values is a crucial first point. By normalizing the data, we bring pixel values to a common scale. This reduction in pixel value variation across different images allows the model to better interpret features and converge more quickly during training. Moreover stability in learning is also greatly improved through data normalization. By limiting the range of pixel values, we prevent the model from being disrupted by extreme values, which could compromise the optimization process and lead to convergence issues [1]. Another significant advancement is the improvement in model

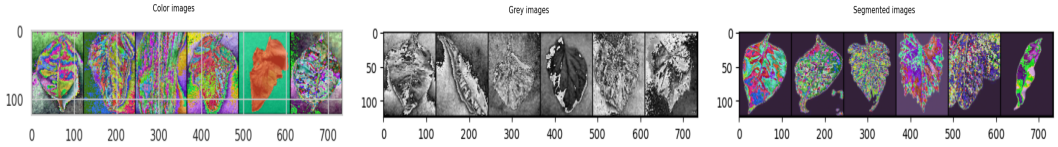

Figure 2: Image types after applying data augmentation techniques and normalization.

generalization. Normalizing the data reduces the model's sensitivity to variations in pixel values across images. This adaptation enables it to generalize more effectively, meaning it performs well on data it hasn't seen during training. Finally, preventing overfitting is another crucial point. Data normalization helps limit the model's capacity to memorize specific features of the training data, encouraging the learning of more general features and better generalization on new data.

For calculating of mean (μ) and standard deviation (σ) for RGB images (and segmented images):

$\texttt{mean}\ (\mu_i): \mu_i = \frac{\sum_{j=1}^{n} I_{ij}}{N}$; $\texttt{standard deviation}\ (\sigma_i): \sigma_i = \sqrt{\frac{\sum_{j=1}^{n}(I_{ij}-\mu_i)^2}{N}}$ where I_{ij} represents the value of pixel j in channel i of the image, N is the total number of pixels in the set of images, and n is the total number of images.

For calculating of mean (μ) and standard deviation (σ) for grayscale images:

$\texttt{mean}\ (\mu): \mu = \frac{\sum_{j=1}^{n} I_j}{N}$; $\texttt{standard deviation}\ (\sigma): \sigma = \sqrt{\frac{\sum_{j=1}^{n}(I_j-\mu)^2}{N}}$ where I_j represents the value of pixel in a channel of the image, N is the total number of pixels in the set of images, and n is the total number of images.

These formulas allow for the calculation of necessary statistics (means and standard deviations) to normalize image data, which is crucial for preparing data for training the neural network model. One can view the some resulting images on a figure 2 after applying normalization and data augmentation techniques (random vertical/horizontal tilting of the image with a probability of 0.5, randomly adjusts brightness, contrast, saturation and hue).We will divide our dataset into three sets: a training set (65%), a validation set (15%), and a test set (20%) for building our AI-based image recognition system. Each of these sets will have a batch size of 64, selected based on the computational capabilities and memory constraints of the hardware.

2. Evaluation Protocol

Before describing the model architecture, we outline the evaluation protocol used to measure the model's performance. This protocol is based on the definition of the loss function and accuracy [6]. The loss function we use is cross-entropy loss, which is commonly employed for classification tasks. It measures the difference between the predicted probability distribution and the actual label distribution. The formula for the cross-entropy loss is:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij})$$

where N is the total number of samples, C is the number of classes, y_{ij} is the true label (1 if sample i belongs to class j, otherwise 0), and $\hat{y}_{ij}$ is the predicted probability that sample i belongs

to class j. Accuracy, another metric commonly used to evaluate classification models, is defined as the ratio of correct predictions to the total number of predictions. The formula for accuracy is:

$$\text{Accuracy} = \frac{\sum_{i=1}^{N} \mathbb{1}(\hat{y}_i = y_i)}{N}$$

where N is the total number of samples, $\hat{y}_i$ is the predicted class for sample i, y_i is the actual class for sample i, and $\mathbb{1}(\cdot)$ is the indicator function that is 1 if the argument is true and 0 otherwise. By using these two metrics, we can evaluate the model's performance in terms of both the accuracy of the predictions and the quality of the match between the predicted probabilities and the actual labels.

3. Model Architecture and Training

i. Reducing Model Size and Applying Regularization

To build a successful deep learning model while avoiding overfitting and underfitting, we implemented two key techniques: reducing the size of the architecture and applying regularization.

For the reduction of the network size, we began with a 'naive' model comprising a single convolutional layer and a fully connected network [4]. In this process, we systematically adjusted parameters such as the number of neurons, filters, and other architectural elements like padding and stride.

By combining these strategies, we aimed to strike a balance between model complexity and generalization performance, ultimately enhancing the robustness and effectiveness of our deep learning model.

ii. Weight Initialization

Weight initialization methods, such as Kaiming He and Xavier [3], are crucial in the design of neural networks. The Kaiming He method is primarily used for convolutional layers with ReLU activation, aiming to maintain a constant activation variance. It initializes weights according to a normal distribution centered at zero with a standard deviation calculated based on the number of inputs to the layer. The weight initialization formula for the Kaiming He method is given by:

$$\text{Weights} \sim \mathcal{N}(0, \sqrt{\frac{2}{n}})$$

where n is the number of inputs to the layer.

On the other hand, the Xavier method is more suited for fully connected layers and aims to maintain a constant activation variance throughout the network. Weights are initialized according to a normal distribution with a standard deviation calculated based on both the number of inputs and outputs of the layer. The weight initialization formula for the Xavier method is given by:

$$\text{Weights} \sim \mathcal{N}(0, \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}})$$

where n_{in} is the number of inputs and n_{out} is the number of outputs of the layer.

These initialization techniques are essential for effectively starting the training of neural networks, promoting convergence, and improving model performance.

iii. Description of Deep Learning Architecture

Our CNN architecture comprises several convolutional layers followed by max-pooling layers for feature extraction. The number of input channels is determined based on the type of input images. For RGB and segmented images, the network is initialized with 3 input channels, while for grayscale , it is initialized with 1 input channel.

The convolutional layers are configured with kernel size 3x3, stride 1, and padding 1 to maintain spatial dimensions. The first convolutional layer ('conv1') takes input images and outputs feature maps with 32 channels. Subsequent convolutional layers ('conv2', 'conv3', 'conv4') increase the number of feature channels to 64, 128, and 256, respectively, capturing increasingly complex features.

After each convolutional layer, we apply the ReLU activation function to introduce non-linearity and enhance feature representation. Following the convolutional layers, max-pooling layers downsample feature maps to reduce spatial dimensions and extract dominant features.

he output of the convolutional layers is then flattened and passed through fully connected layers (fc1, fc2, fc3) for classification. The first fully connected layer (fc1) has 512 neurons, followed by a layer with 64 neurons (fc2), and finally, an output layer with 38 neurons representing the number of classes. These dimensions are chosen based on considerations such as the size of the input images and the complexity of the classification task. At the output layer, the softmax function is applied to convert the raw scores into a probability distribution over the classes, facilitating the interpretation of the network's output as class probabilities. During the forward pass, input images are passed through the convolutional and pooling layers, followed by flattening and classification through the fully connected layers. The final layer produces class predictions for crop disease identification. Hence the model has `6 846 758` trainable parameters.

iv. Choosing Learning Rate and Optimizer:

Finding the right learning rate for training the model is an ongoing area of research where a lot of research has been made. PyTorch provides some techniques to tune the learning rate, which are provided in the `torch.optim.lr_scheduler` package: `StepLR`, `MultiStepLR`, `ExponentialLR`, and `ReduceLROnPlateau` [7]. In this project, we will rely on grid search by defining the learning rate values as 10^{-2}, 10^{-3}, and 10^{-4}, observing the evolution of loss and accuracy to make relevant choices for optimizing our model. We observe from the figure that the learning rate value of 10^{-4} is not interesting at all because the loss remains around 3.1 both during training and validation. The other values are satisfactory, but the most interesting one is 10^{-3}, which allows the loss function to converge towards 0 a little faster after 20 iterations (figure 1). Therefore, 10^{-3} is the chosen learning rate value.

As for the choice of optimizer, we tested three optimizers: RMSProp, Adam, and SGD. SGD offers very poor results because the loss decreases very slowly and seems to plateau around 3 after 20 iterations. While RMSProp allows the loss function to decrease and the accuracy to increase during both training and validation, its unstable behavior during validation may pose some issues. On the

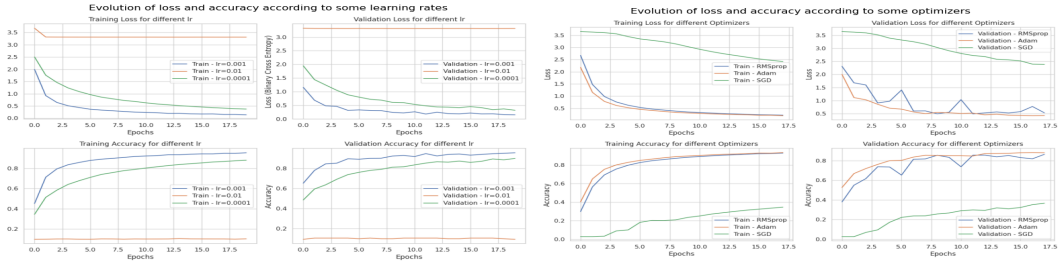

Figure 3: Evolution of loss and accuracy according to some learning rates and optimizers

other hand, Adam shows the loss function decreasing towards 0 after 20 iterations, and an accuracy of over `90%` and a loss value around `0.26`. Therefore, the choice of Adam as the optimizer seems to be the most appropriate in this case, as it shows a decrease in the loss function towards 0 after 20 iterations and an accuracy of over 90% (figure 1). This suggests that Adam is capable of effectively optimizing the model's parameters and producing good performance in terms of accuracy.

4. Training the Model on Training, Validation, and Test Sets

After defining the architecture of our convolutional neural network and optimizing it based on hyperparameters such as learning rate and optimizer, we can now proceed to train, validate, and test the model to evaluate its performance. The process is as follows: training on the training set and testing on the validation set to select the best hyperparameters. Subsequently, the model is defined based on these hyperparameters, trained on the combined training and validation sets, and tested on the test set [2].

In this project, the training of our CNN model was conducted using GPU resources to ensure efficient and timely processing of the data. Utilizing GPU capabilities enabled us to achieve high accuracy and low loss values within a reasonable training time. The model's performance is evaluated by plotting the evolution of loss and accuracy over 20 iterations for the training, validation, and test sets across the three versions of the dataset (grayscale, color, and segmented). The analysis shows that our model consistently achieves over 90% accuracy across all datasets, with a loss value around 0.26 after 20 iterations. This demonstrates the model's robust performance and ability to generalize well to different types of image data(figure 4).

5. Analysis of model performance and criticisms

Analyzing the performance across grayscale, color, and segmented datasets provides valuable insights into the classification capabilities of the model. Across all datasets, certain classes consistently exhibit high success rates, indicating robust classification regardless of the image type. Notably, classes like `Soybean__healthy` and `Orange__Haunglongbing_(Citrus_greening)` maintain consistently high success rates, suggesting that the model performs well for these classes irrespective of color information (figure 5).

On the other hand, differences in performance between grayscale and color datasets are evident for specific classes. Classes such as `Tomato__Early_blight` and `Apple__Cedar_apple_rust` demonstrate lower success rates in grayscale compared to color datasets, indicating that color informa-

Figure 4: Evolution of accuracy and loss on the test set for the three data set versions.

tion may play a crucial role in distinguishing these diseases. Despite overall high success rates, certain classes consistently present challenges for accurate classification. `Tomato__Late_blight` and `Tomato__Septoria_leaf_spot`, for instance, consistently display lower success rates across all datasets, highlighting potential complexities in distinguishing these diseases from others (figure 6).

The model demonstrates robust performance for several classes across different datasets, suggesting its ability to generalize well to diverse image types and conditions. However, there are still opportunities for improvement, particularly for classes with lower success rates. Further analysis and refinement of the model may enhance its performance and address the challenges observed in classification.

6. Conclusion

In this project, we developed a convolutional neural network (CNN) architecture with four convolutional layers, inspired by the methodology outlined in "Deep Learning with PyTorch" [7], to identify crop diseases from smartphone-captured images. While our model demonstrated overall strong performance, it faced several challenges, including an inability to detect certain crop diseases and imbalanced classes within our dataset, potentially influencing the results.

To enhance our model, future research should focus on several aspects. Firstly, a deeper exploration of imbalanced data, along with the application of techniques such as oversampling or undersampling, could improve its performance on underrepresented classes. Additionally, further adjustments to the CNN architecture, such as adding extra layers or modifying activation functions, could also prove beneficial for disease detection.

Furthermore, expanding our dataset beyond the current 54,000 images, by including a greater variety of crop diseases and conditions, is crucial to bolstering the robustness and generalization of the model. It may also be worthwhile to consider continuing our data collection efforts by capturing images across a wider range of geographical regions and weather conditions, enabling our model to better generalize the characteristics of different crop diseases. By addressing these challenges and refining our approach, we aim to provide more accurate and reliable tools for crop disease management, thereby contributing to agricultural sustainability and food security initiatives.

[0]For more details on the results obtained in this project, please visit the following GitHub repository: https://github.com/mvom/https-github.com-mvom-dlCropDiseaseImageClassification.git

Appendix

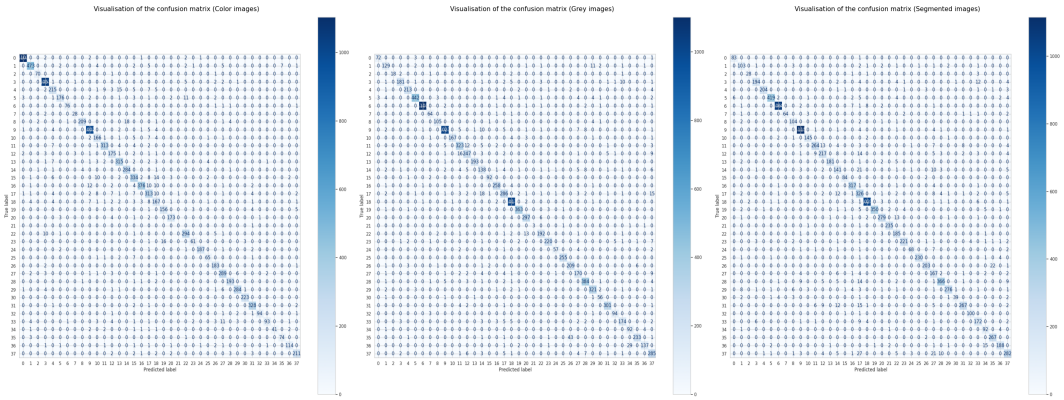

Figure 5: Visualisation of the confusion matrix for each three data set versions on a test set.

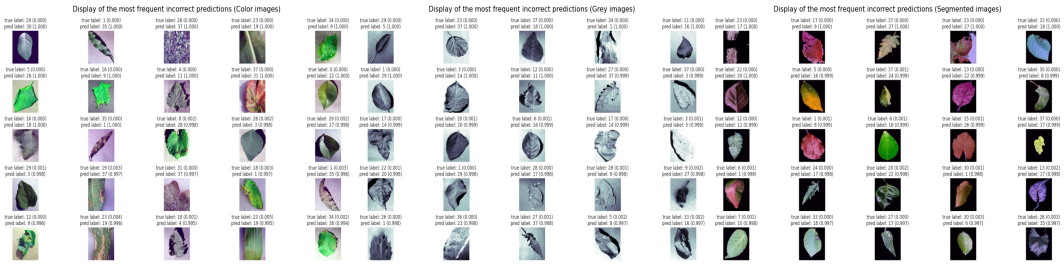

Figure 6: Display of the most frequent incorrect predictions for each three data set versions on a test set.

Bibliography

[1] Saleh Albelwi and Ausif Mahmood. A framework for designing the architectures of deep convolutional neural networks. Entropy, 19(6):242, 2017.

[2] Bart Bogaerts, Gianluca Bontempi, Pierre Geurts, Nick Harley, Bertrand Lebichot, Tom Lenaerts, and Gilles Louppe. Artificial intelligence and machine learning. 2021.

[3] Xiaofeng Ding, Hongfei Yang, Raymond H Chan, Hui Hu, Yaxin Peng, and Tieyong Zeng. A new initialization method for neural networks with weight sharing. In Mathematical Methods in Image Processing and Inverse Problems: IPIP 2018, Beijing, China, April 21–24, pages 165–179. Springer, 2021.

[4] Isma Hadji and Richard P Wildes. What do we understand about convolutional networks? arXiv preprint arXiv:1803.08834, 2018.

[5] David Hughes, Marcel Salathé, et al. An open access repository of images on plant health to enable the development of mobile disease diagnostics. arXiv preprint arXiv:1511.08060, 2015.

[6] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. arXiv preprint arXiv:1702.05659, 2017.

[7] Vishnu Subramanian. Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch. Packt Publishing Ltd, 2018.