Optimizing Plant Disease Classification Models: A Comparative Study of Custom, ResNet 18, and EfficientNet B1 Architectures with Hyperparameter Analysis

Thi Kim Oanh Nguyen
a1879781

1. Introduction and Background

1.1. Problem Statement

The early detection of disease symptoms in plants represents a significant milestone for both the crop protection industry and addressing key societal and environmental challenges. These challenges include the need to increase crop yields or reduce yield losses to ensure food security for a projected global population of 10 billion by 2050 [6]. Ul Haq and Ijaz (2020) [7] categorizes the assessment of plant diseases into three fundamental aspects: the measurement of disease incidence, disease severity, and yield loss. Incidence is defined as the proportion of the plant that shows signs of disease. Severity is quantified as the portion of the plant area affected by infection, while yield loss refers to the portion of the crop harvest that is destroyed or negatively impacts the quality of the produce. Although disease severity and yield losses hold greater significance for farmers, measuring disease incidence is considerably more challenging and, in certain situations, impossible until it is too late. Visual assessment of plant health and the identification of specific diseases have traditionally relied on human expertise, which can be time-consuming, subjective, and limited in scalability. Furthermore, the severity of diseases often remains unnoticed until visible symptoms manifest, making it challenging to take timely corrective measures.

In response to these agricultural challenges, there is a growing need for automated and accurate disease detection systems. In this research, we create a specialized Convolutional Neural Network (CNN) for plant disease classification, using the PlantVillage dataset. The overarching question guiding this research is: Can a carefully designed, handcrafted CNN outperform or match the performance of well-established pretrained models such as ResNet and EfficientNet when it comes to diagnosing plant diseases? Our specific objectives include the development of a robust CNN, benchmarking it against pretrained models, and drawing insights into the advantages and limitations of each approach.

About methodology, we address the challenges of preparing raw images for model training, including resizing, data augmentation, and normalization, to enhance the quality and consistency of the dataset. Additionally, we explore various hyperparameters and fine-tune different variants of ResNet and EfficientNet models, with an emphasis on best practices for optimizing performance. Simultaneously, a custom model will be carefully trained to ensure comparability with ResNet and EfficientNet models.

The impact of our research could be significant on a global scale. It has the potential to simplify the detection of plant diseases, leading to increased crop yields and promoting the long-term sustainability of food production. Moreover, our study showcases how machine learning can effectively address real-world challenges, extending its relevance beyond agriculture.

This paper proceeds as follows: Section 2 describes the method, Section 4 presents the experimental results, and Section 5 discusses future directions of our research.

1.2. Related Work

Some research papers have also employed the PlantVillage dataset for plant disease prediction. Mohanty et al. (2016) [5] applied transfer learning with the Stochastic Gradient Descent optimizer and ran for a total of 30 epochs, choosing a batch size of 24. They achieved their best performance of 99.34% using GoogLeNet. Similarly, Atila et al. (2021) [1] compared different Efficientnet models, and the highest accuracy score of 99.91% was achieved using the B5 model with a batch size of 16 and the Adam optimizer. The most remarkable achievement was the application of an adaptive minimal ensembling technique, with an exceptional accuracy score of 100% (Bruno et al., 2020)[5]. This approach involved the training of two EfficientNet-b0 weak models, which were subsequently integrated to create a novel combined classifier, resulting in peak performance.

In this study, we have developed a custom CNN model and demonstrated its comparable or even superior performance to two well-known pre-trained models, ResNet 18 and EfficientNet B1. Unlike prior work, such as Mohanty et al. (2016)[5], which did not provide a comprehensive comparison of these two methods, we go into the details of how we constructed our custom model and contrasted various parameters in the initial state.

As a result of our experiments, we have achieved promising outcomes. Our custom model outperformed all other models with an accuracy of 99.6686%, requiring more training time. On the other hand, ResNet 18 and EfficientNet B1 achieved accuracies of 97.0539% and 99.5397%, respectively, while demonstrating more stability than the custom model. This demonstrates the potential for early detection of plant diseases with more accurate results after being experimented by each model.

2. Method Description

2.1. Basic CNN Components

Convolutional Layers

The primary purpose of the convolutional layers is to extract relevant features from the input data. For image data, these layers help identify patterns such as edges, corners, and textures. They consist of multiple learnable filters or kernels. Each kernel is a small grid-like matrix. These kernels are applied to local regions of the input data to detect specific patterns. The operation performed by a kernel is called convolution.

Each neuron in a feature map (output of a convolutional layer) is connected to a local region in the input data known as its receptive field. The size of this receptive field is determined by the dimensions of the convolution kernel. One critical aspect of convolutional layers is weight sharing. The same set of weights (kernel) is used for all spatial locations within the receptive field. This allows the network to detect the same feature in different parts of the input.

The output of each convolutional layer is a set of feature maps, also known as activation maps. Each feature map represents the response of a specific kernel to different local patterns in the input. These feature maps collectively capture various features at different spatial locations. To introduce nonlinearity into the model, an element-wise nonlinear activation function, such as ReLU (Rectified Linear Unit), is applied to the convolved results for each feature map. This step helps the network capture complex patterns and relationships. CNNs often consist of multiple convolutional layers stacked on top of each other. These layers gradually learn to recognize higher-level features by combining low-level patterns identified in earlier layers.

Pooling Layers

Pooling layers down-sample the spatial dimensions of feature maps produced by the previous convolutional layers, reducing computational complexity while enhancing translation invariance. They help the network focus on the most important features and contribute to the success of CNNs in tasks such as image recognition and object detection. Max pooling and average pooling are the two common operations used. In max pooling, for each local region in a feature map, the maximum value is retained, capturing the most important feature in that region. In average pooling, the average value of the elements in the local region is calculated. This operation is less sensitive to outliers.

Like convolutional layers, each neuron in a pooling layer is also connected to a local receptive field. The size of this field is determined by the dimensions of the pooling kernel. Similar to convolution, pooling is performed with a specified stride. The stride determines how much the pooling window moves as it scans the feature map.

Pooling layers typically operate on each feature map independently. For each feature map, a separate pooling operation is performed, generating new feature maps with reduced spatial dimensions. Pooling layers are often inserted between convolutional layers in a CNN architecture. Stacking convolutional layers with pooling layers allows the network to capture hierarchical features from low-level to high-level abstractions.

Fully-connected Layers

Fully-connected layers are located at the end of a CNN architecture. Unlike convolutional and pooling layers, where operations are performed on local regions of the input, each neuron in a fully-connected layer receives input from all neurons in the preceding layer and feeds its output to all neurons in the subsequent layer. To connect a fully-connected layer to the output of preceding convolutional or pooling layers, the feature maps from the preceding layers are typically flattened into a one-dimensional vector. This vector is then used as input to the fully-connected layer.

Each connection between neurons in fully-connected layers has its associated weight and bias parameters. During training, the neural network learns these parameters through backpropagation and optimization algorithms. The weights determine the strength of connections between neurons, while biases allow neurons to have some degree of independence from each other.

The final fully-connected layer in a CNN is often followed by an activation function, such as softmax, which converts the network's output into a probability distribution over the possible classes. While convolutional and pooling layers capture local and spatial features, fully-connected layers abstract away spatial information, focusing on high-level semantics and patterns. This abstraction makes them suitable for tasks like classification.

Activation Function

Without activation functions, the entire neural network would behave like a linear regression model, and it would not be able to capture complex patterns in the data. Activation functions are applied element-wise to the output of

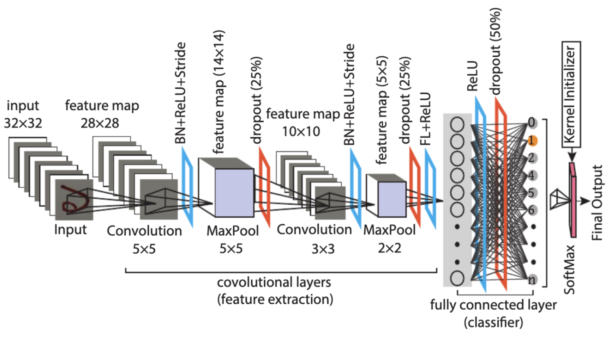

Figure 1. CNN model[3]

each neuron in the CNN layers, including convolutional and fully-connected layers. This introduces non-linearity and allows the network to learn complex hierarchical features from the data. ReLU is one of the most popular activation functions. It replaces negative input values with zero and leaves positive values unchanged. ReLU is computationally efficient and helps mitigate the vanishing gradient problem. Leaky ReLU is a variant of ReLU that allows a small, non-zero gradient for negative inputs, preventing ”dying ReLU” problems.

Loss Function

For multi-class classification tasks, categorical cross-entropy loss is used. It measures the dissimilarity between predicted class probabilities and the true class labels. It is typically employed in softmax-based output layers. Hinge loss is often used in support vector machines (SVMs) and is suitable for classification tasks.

During training, the CNN uses optimization techniques like gradient descent to minimize the chosen loss function. The network's weights and biases are iteratively adjusted to reduce the loss. In some cases, regularization terms, such as L1 or L2 regularization, can be added to the loss function to prevent overfitting and encourage the network to have smaller weights.

Optimization

Gradient descent is a widely used optimization algorithm in CNNs. It's an iterative method that updates the parameters in a way that reduces the loss function. The key idea is to move in the direction where the loss decreases the most. The optimization process typically starts with an initial set of parameters, often initialized randomly or with pre-trained values. During training, data is fed forward through the network. The network makes predictions based on the current parameter values. The loss function is calculated to quantify how far off the predictions are from the true target values. Backpropagation computes the gradients of the loss function with respect to each parameter. Gradients indicate how much each parameter should be adjusted to decrease the loss.

The most common one is Gradient Descent, but there are variations and improvements like Stochastic Gradient Descent or Adam. SGD uses a fixed learning rate, while Adam adapts the learning rate for each parameter based on past gradient information. Adam's updates are more stable and less noisy due to its adaptive learning rate. This often leads to faster convergence compared to vanilla SGD. SGD can incorporate momentum to smooth updates, but Adam already includes momentum-like elements as part of its update rule.

Batch Normalization

Batch normalization works by normalizing the input of each layer within a mini-batch during training. Given a mini-batch of activations, the normalization step calculates the mean (μ) and variance (σ^2) for each feature:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{1}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2 \tag{2}$$

x_i represents the activation values in the mini-batch, and m is the number of instances in the mini-batch.

Using the mean and variance calculated above, Batch Normalization normalizes the input with the following equation:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{3}$$

$\hat{x}_i$ represents the normalized activation and ϵ is a small constant added to the variance for numerical stability.

Batch Normalization then scales and shifts the normalized activations by learned parameters γ and β:

$$y_i = \gamma\hat{x}_i + \beta \tag{4}$$

y_i represents the output after scaling and shifting.

During training, γ and β are learned by the model through backpropagation. During inference, the model utilizes a running average of the mean and variance computed during training to normalize the input, ensuring consistent behavior between training and inference.

2.2. ResNet Model

ResNet, short for ”Residual Networks,” is a deep convolutional neural network (CNN) architecture that was introduced in 2015. ResNet is known for its ability to train very deep neural networks effectively, overcoming the problem of vanishing gradients that can occur in traditional deep networks.

The fundamental idea behind ResNet is the use of residual blocks, also known as residual connections or skip connections. These blocks enable the training of very deep networks by allowing information to flow easily through the network, mitigating the vanishing gradient problem.

In a residual block, the input to the block (usually referred to as the identity) is combined with the output of one or more convolutional layers to form the final output of the block. This combination is done using element-wise addition. Mathematically, a residual block can be represented as:

$$y = F(x, W) + x \tag{5}$$

where:

y is the output of the block.

x is the input to the block.

F(x,W) represents the transformation performed by the convolutional layers (parameterized by W).

The addition of x is what makes this a "residual" block.

ResNet comes in several variants, including ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, which differ in the number of layers and model complexity.

2.3. EfficientNet Model

EfficientNet is designed to provide a balance between model size, computational resources, and performance. It was introduced in 2019. The primary idea behind EfficientNet is to scale the model's depth, width, and resolution in a systematic and balanced manner. Traditionally, increasing one of these factors would require proportionally increasing computational resources, making models larger and more computationally intensive. EfficientNet introduces a novel compound scaling method to simultaneously scale all these factors to achieve better performance.

EfficientNet uses three scaling dimensions. EfficientNet scales the depth of the network by using a compound coefficient "d," which is user-defined. For example, if you specify d=1.2, the network will have 1.2 times more layers than the base model. It scales the model's width or the number of channels in each layer with the compound coefficient "w". Larger "w" values increase the number of filters in convolutional layers. It scales the input image resolution, which affects the size of the feature maps by the compound coefficient "r".

The compound scaling method combines these three dimensions to create a compound scaling coefficient "phi" using a formula:

$$\phi = \alpha^d \cdot \beta^w \cdot \gamma^r \tag{6}$$

where:

α, β, and γ are user-defined hyperparameters that control the scaling of depth, width, and resolution.

d, w, and r are the compound coefficients for depth, width, and resolution scaling.

EfficientNet optimizes this compound coefficient to achieve the best trade-off between model size and performance. The number of layers and the architecture's design depend on the chosen compound coefficient "phi" and the specific model variant (B0, B1, B2, ..., B7).

3. Method Implementation

Our codes are here: https://github.com/OanhOlivia/Deep-Learning/blob/main/a1879781_DL_A2.ipynb

4. Experiment and Analysis

4.1. Data Description and Data Augmentation

The PlantVillage dataset contains 54,305 high-quality images of 14 different plant species (Apple, Blueberry, Cherry, Corn, Grape, Orange, Peach, Bell Pepper, Potato, Raspberry, Soybean, Squash, Strawberry, and Tomato). It is categorized into 38 classes with 12 healthy and 26 diseased plant variations, including 17 fungal diseases, 4 bacterial diseases, 2 mold (Oomycete) diseases, 2 viral diseases, and 1 disease caused by a mite (Bruno et al., 2020)[5].

To prepare the raw images for model training, we randomly flips images horizontally and vertically with a 50% chance to introduce variety into the training data by providing the model with mirrored versions of the images. Color jitter is applied with a 10% probability. It randomly adjusts the brightness, contrast, saturation, and hue of the images to help the model become more resilient to variations in lighting conditions and object appearances. The images are resized to a common size of 300x300 pixels and center-cropped to a final size of 260x260 pixels. The data is split into training (80% of the dataset), validation (10% of the dataset), and test data (10% of the dataset).

4.2. Performance Comparison from Hyperparameter Selection

In this research, we will implement one custom model, ResNet 18, and EfficientNet B0, B1, B2. EfficientNet, known for its outstanding accuracy scores, holds great potential. We aim to determine the highest accuracy score we can get when using EfficientNet for the PlantVillage Dataset. Prior to utilizing pre-trained models, we will train a basic CNN from scratch. This will help us explore whether a custom model can surpass the performance of pre-trained models. We'll also delve into fine-tuning different hyperparameters during the model design phase to identify the optimal settings that contribute to a high-performing CNN model. Furthermore, due to its fast training times and convenience, we will use ResNet 18 for performance comparisons with varying batch sizes and optimizers to choose the most effective ones. Finally, we will compare different EfficientNet models to select the most suitable one for our plant disease classification task.

4.2.1 Image Sizes

PlantVillage is a dataset of plant disease images, and accurate disease diagnosis often requires capturing fine details.

Batch Size	Test Loss	Test Accuracy
32	**0.8928**	**97.0539**
128	0.9039	96.5568
256	0.9234	95.9676

Table 1. Test Accuracy with different Batch Sizes

While a larger image size can help preserve these details, it can also increase computational demands. Choosing a good image size is a good starting point for the custom model and be used effectively with pre-trained models as well. ResNet models are relatively flexible when it comes to input image size. They are often used with a variety of image sizes, including 224x224, 256x256, and 299x299 pixels. EfficientNet models are known for their efficiency in terms of model size and computational resources. Some recommended image sizes for EfficientNet include 224x224, 260x260, and 300x300 pixels.

In this research, we decided to resize images to 300 pixels and then applying center cropping to reduce them to 260 pixels. A 300x300 size provides a good balance between capturing image details and keeping the image size manageable. Center cropping ensures that the most relevant and informative part of the image is retained. This approach keeps the images relatively large (260x260), which is beneficial for capturing fine details, while also maintaining a consistent aspect ratio.

4.2.2 Batch Sizes

The batch size determines how many data samples are processed simultaneously during each forward and backward pass. In the context of GPU memory, the batch size should be chosen so that it fits within the available memory. A smaller batch size allows the model to update its weights more frequently during training, which can result in faster convergence and better generalization. It can also help the model escape local minima. Larger batch sizes generally lead to noisier weight updates, as they involve more samples in each update. This can slow down the training process and may result in a less optimal solution [4].

In our research, batch size 32 seems to be the most suitable for training ResNet 18 on the PlantVillage Dataset. The batch size of 32 performed best because it struck a balance between frequent weight updates and computational efficiency. It allowed the model to find a better solution with a lower test loss and higher test accuracy, as illustrated in Figure 2 and Table 1. The other batch sizes (128 and 256) had less desirable outcomes in training due to their noisier updates or slower convergence. We will use a batch size of 32 for all models in this research.

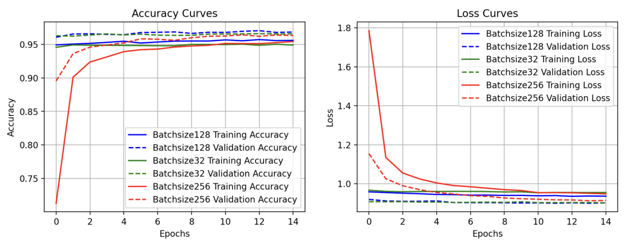

Figure 2. Learning Curves of ResNet 18 with different Batch Sizes

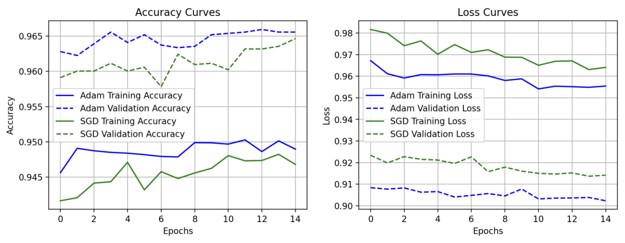

Figure 3. Learning Curves of ResNet 18 with Adam and SGD optimizer

Optimizer	Test Loss	Test Accuracy
Adam	**0.8928**	**97.0539**
SGD	0.9069	96.7962

Table 2. Test Accuracy with different Optimizers

4.2.3 Optimizer

The Adam optimizer is known for its adaptive learning rates, which make it well-suited for a wide range of problems. In our research, it seems that Adam's adaptive learning rate mechanisms have allowed our model to converge faster and achieve a lower test loss and a higher test accuracy (Figure 3 and Table 2). On the other hand, the Stochastic Gradient Descent (SGD) optimizer might not have adapted to the dataset as efficiently as Adam. It has a fixed learning rate and is more sensitive to hyperparameter tuning. The slightly higher test loss and lower test accuracy (96.7962%) suggest that the model trained with SGD might not have reached the same level of performance as the one trained with Adam. In summary, the Adam optimizer appears to provide better results for PlantVillage Dataset, likely due to its adaptive learning rate mechanisms, which can lead to faster convergence and improved model performance. We will use Adam optimizer for all models in this research.

4.2.4 Custom Model Design

Activation Function

ReLU (Rectified Linear Unit) is computationally effi-

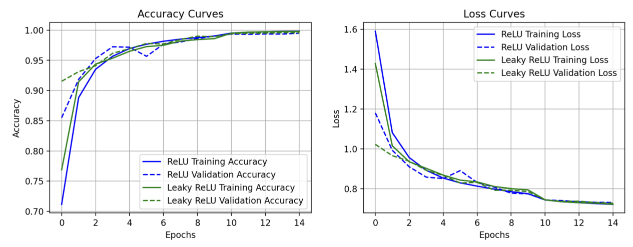

Figure 4. Learning Curves of custom model with ReLU and Leaky ReLU

Activation Function	Test Loss	Test Accuracy
ReLU	0.7312	99.3740
Leaky ReLU	**0.7314**	**99.6317**

Table 3. Test Accuracy with different Activation Functions

cient and helps mitigate the vanishing gradient problem, but it can sometimes lead to a phenomenon known as ”dying ReLU”. This means that some neurons can become inactive, causing the model to lose some of its learning capacity. Leaky ReLU is a variation of ReLU that addresses the dying ReLU problem. It allows a small gradient when the input is less than zero, preventing neurons from becoming completely inactive. As a result from Figure 4 and Table 3, the model using Leaky ReLU achieved slightly higher test accuracy compared to ReLU. We will use Leaky ReLU for our custom model.

Batch Normalization and Drop-out

Without batch normalization and drop-out as Figure 5, we observed that sometimes the training score is higher, and sometimes the validation score is higher. This indicates that the model's performance can be sensitive to the initial conditions and random variations during training. It is more likely to overfit the training data, resulting in fluctuations in the training and validation scores.

Conversely, in Figure 6, batch normalization stabilizes the training process by reducing internal covariate shift, making the model more robust. This helps ensure that the statistics of each layer remain consistent during training, which can lead to a more stable learning process. Additionally, dropout is applied as a regularization technique to prevent overfitting in cases where the model is more complex or prone to overfitting. However, batch normalization introduces additional scaling and shifting parameters for each layer, which can increase the test loss slightly (Table 4). While it may slightly increase the test loss, adding batch normalization is a trade-off for improved generalization and model stability.

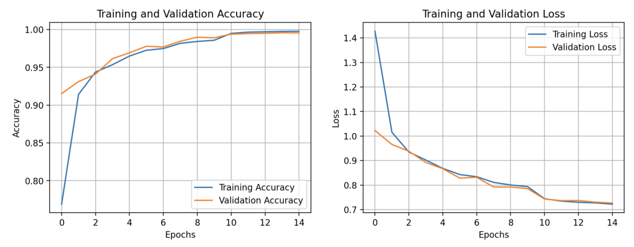

Figure 5. Learning Curve without Batch Normalization and Drop-out

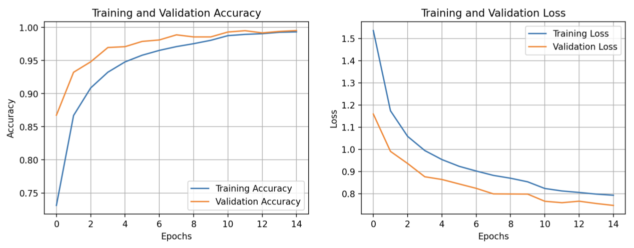

Figure 6. Learning Curve with Batch Normalization and Drop-out

Batch Normalization -Dropout	Test Loss	Test Accuracy
Yes	0.7435	**99.6686**
No	**0.7314**	99.6317

Table 4. Test Accuracy with Batch Normalization and Drop-out

4.3. Performance Comparison of Custom Model, ResNet, and EfficientNet Implementations

4.3.1 Performance Comparison between different EfficientNet Models

EfficientNet B0 is the base model in the EfficientNet series. It's relatively smaller and less complex compared to B1 and B2. While it still performs well, it might have slightly more limitations in capturing complex patterns and features in the PlantVillage Dataset due to its smaller architecture. EfficientNet B1 is a step up from B0 with a slightly larger architecture. It has more capacity to capture intricate patterns and features within the dataset. This increases in capacity might have contributed to its improved accuracy and slightly lower loss compared to B0. EfficientNet B2 is larger and more complex than B0 and B1. While it has more capacity, it did not perform as well as B1 in this specific case. It's possible that the increased complexity led to more challenges in training, resulting in a slightly higher test loss and slightly lower accuracy.

EfficientNet B1 achieved the best result (Figure 7 and Table 5) because it strikes a good balance between model complexity and performance. It offers more capacity to capture relevant features compared to B0 without becoming

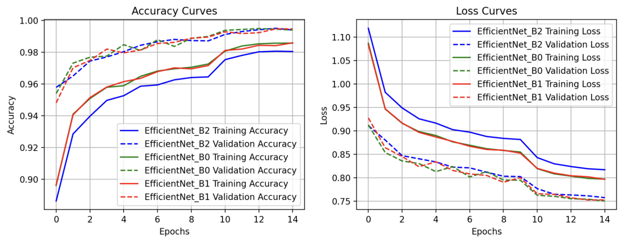

Figure 7. Learning Curves of different EfficientNet Models

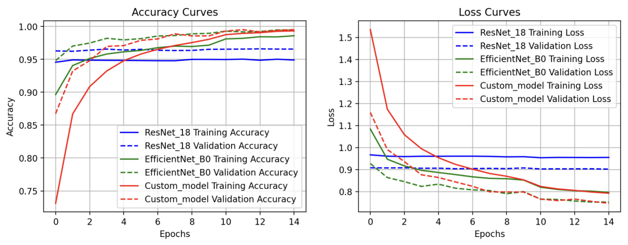

Figure 8. Learing Curves of different Models

EfficientNet	Test Loss	Test Accuracy
B0	0.7566	99.3187
B1	**0.7520**	**99.5397**
B2	0.7662	99.4108

Table 5. Test Accuracy of different EfficientNet Models

overly complex like B2. The PlantVillage Dataset is extensive with 53,000 images and 38 classes. B1's architecture seems to be well-suited to handle the complexity and variety in this dataset, leading to the best overall test accuracy and a competitive test loss.

4.3.2 Performance Comparison between Custom Model, ResNet 18 and EfficientNet B1

ResNet 18 has a total of 11,196,006 parameters, 11,176,512 of which are non-trainable. It consists of multiple building blocks, specifically "BasicBlock," which are repeated to create a deep network. These building blocks include two or three convolutional layers with batch normalization and ReLU activation for feature extraction, followed by a max-pooling layer for spatial reduction. The blocks are repeated to create a hierarchy of features, starting with 64 channels and gradually increasing to 512 channels. It concludes with an adaptive average pooling layer that reduces the spatial dimensions to 1x1 and a fully connected layer with 38 output units. The model uses batch normalization to improve training stability and dropout to prevent overfitting.

EfficientNet B1 has a total of 9,561,402 parameters, 6,860,052 of which are non-trainable. There are multiple convolutional layers with varying numbers of output channels (e.g., 32, 16, 64, 32, 128, 48, 192, 96, 384, 112, 672, 192, 1152, etc.). Batch normalization is applied after some of the convolutional layers. It is used to normalize the activations of the previous layer, helping with training stability. SiLU (Sigmoid-weighted Linear Unit) activation functions are applied after some of the convolutional layers to introduce non-linearity. Squeeze and Excite blocks are used to recalibrate the feature maps. They include operations like convolution, activation (SiLU), and sigmoid to compute scaling factors for each channel. Identity and Residual locks are used to create residual connections within the network, allowing for the flow of information through skip connections.

Custom model has a total of 67,223,142 trainable parameters, with three convolutional layers, three max-pooling layers, two fully connected layers and Leaky ReLU activation function. Unlike ResNet 18 and EfficientNet B1, batch normalization and dropout are specifically applied to the fully connected layer but not to the convolutional layers or other parts of the network in our custom model.

Table 6 shows that our custom model outperformed ResNet 18 and EfficientNet B1 with the lowest test loss and highest test accuracy. Custom model is a simpler model with fewer layers and parameters. Simple models may be easier to optimize during training and generalize better to the test data. If the task doesn't require a very deep or complex architecture, a simpler model might be more efficient and effective. Custom model may have been designed specifically for the task or dataset at hand. It is also easier to perform more extensive hyperparameter tuning to optimize the performance of a custom model. The choice of learning rate, batch size, optimization algorithm, and other hyperparameters can significantly impact model performance. Our custom model has significantly more trainable parameters (67,223,142) compared to ResNet-18 (19,494) and EfficientNet B1 (2,701,350). Having more parameters allows the model to capture more intricate details from the training data, which can be beneficial if the dataset is diverse. It was trained for a longer duration (61 minutes) compared to ResNet-18 (53 minutes) and EfficientNet B1 (48 minutes). More training time can allow the model to converge better and potentially find better solutions.

However, as seen in Figure 8, it can be observed that ResNet-18 and EfficientNet B1 exhibit greater stability compared to the custom model. The custom model displays a slight tendency toward overfitting toward the end of the epoch, where training accuracy is more likely to surpass validation accuracy. Therefore, further research is necessary to identify and address these errors and to advance our research.

Model	Training Time	Test Loss	Test Accuracy
Custom Model	**61m 1s**	**0.7435**	**99.6686**
ResNet 18	53m 39s	0.8928	97.0539
EfficientNet B1	48m 54s	0.7520	99.5397

Table 6. Test Accuracy of different Models

5. Reflection on Project

5.1. Conclusion

In this study, we conducted a comprehensive examination of various hyperparameters across three distinct model architectures: the Custom Model, ResNet 18, and EfficientNet B1. These models were evaluated in the context of plant disease classification, involving a dataset comprising 53,000 color images distributed among 38 unique classes. Our findings have illuminated the relative strengths and weaknesses of these models, including different variants, providing valuable insights for model selection and development.

First, our study underscores the importance of thoughtful choices in image size, batch size, and optimizer. An image size of 260x260, achieved through center cropping, proves beneficial for capturing fine details while maintaining a consistent aspect ratio. A batch size of 32 performed the best, striking a balance between frequent weight updates and computational efficiency. Due to an adaptive learning rate mechanism, the Adam optimizer stands out as the optimal choice. Our custom model demonstrated impressive performance due to the careful selection of a suitable Leaky ReLU activation function and the utilization of batch normalization and dropout techniques.

Second, EfficientNet B1 emerged as the most competitive choice, surpassing B0 and B2 in terms of test accuracy while maintaining a competitive test loss. It strikes a good balance between model complexity and performance.

Finally, we observed that Custom Model, despite its simplicity, remarkably outperformed ResNet 18 and EfficientNet B1. It achieved the lowest test loss and the highest test accuracy. The Custom Model's success can be attributed to its tailored design, which effectively captured the dataset's characteristics, leading to superior generalization. However, it is noteworthy that while Custom Model exhibited outstanding performance, it showed less stable than the others. This emphasizes the need for further research and optimization to enhance its robustness.

5.2. Future Work

Explore advanced data augmentation techniques to improve model generalization. Techniques like CutMix, MixUp, and AutoAugment can artificially increase the dataset's size and diversity, potentially boosting model performance.

Consider employing ensemble methods to combine predictions from multiple models, such as our custom model, ResNet, and EfficientNet. Ensemble techniques can often mitigate individual model biases and yield superior classification results.

Investigate modifications to the custom model's architecture, including experimenting with deeper or wider networks and different activation functions. Optimizing the architecture specifically for the PlantVillage Dataset may lead to further improvements. Continue exploring the hyperparameter space, including learning rates, weight decay, dropout rates, and batch sizes.

Transition from research to practical application by integrating the developed models into real-world scenarios. Deploying the models for real-time image classification in agricultural settings can contribute to plant disease management and crop protection.

Extend the research to more extensive datasets covering various plant diseases, crops, and environmental conditions. The knowledge gained from this work can then be applied to broader agricultural applications.

Investigate the robustness of the models by introducing various forms of noise and perturbations to the input images. Adapting the models to handle noisy or imperfect data is crucial for real-world applications.

References

[1] Atila, Ü, Uçar, M, Akyol, K Uçar, E 2021, 'Plant leaf disease classification using EfficientNet deep learning model', *Ecological Informatics*, vol. 61, p. 101182–.

[2] Bruno, A, Moroni, D, Dainelli, R, Rocchi, L, Morelli, S, Ferrari, E, ... Martinelli, M 2022, 'Improving plant disease classification by adaptive minimal ensembling', Frontiers in Artificial Intelligence, vol. 5, pp. 868926–868926.

[3] Ibrahim, Z, Diah, NM, Azmi, ME, Abdullah, A Zin, NAM 2022, 'Real-Time Mobile Application for Handwritten Digit Recognition Using MobileNet', in Proceedings of the 11th International Conference on Robotics, Vision, Signal Processing and Power Applications, Springer Singapore, Singapore, pp. 1003–1008.

[4] Keskar, NS, Mudigere, D, Nocedal, J, Smelyanskiy, M Tang, PTP 2017, 'On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima', arXiv.org.

[5] Mohanty, SP, Hughes, DP Salathé, M 2016, 'Using Deep Learning for Image-Based Plant Disease Detection', *Frontiers in Plant Science*, vol. 7, pp. 1419–1419.

[6] Ranganathan, J. et al 2018, *How to sustainably feed 10 billion people by 2050, in 21 charts*, World Resources Institute, accessed 24 October 2023, https://www.wri.org/insights/how-sustainably-feed-10-billion-people-2050-21-charts.

[7] Ul Haq, I & Ijaz, S 2020, *Plant Disease Management Strategies for Sustainable Agriculture Through Traditional and Modern Approaches*, 1st Edition 2020, vol. 13, Springer International Publishing AG, Cham.