**Plant disease Prediction using CNN**

**COMPUTER VISION PROJECT**

**CSE, 6TH SEMESTER**

**Group Members:**

|  |  |
| --- | --- |
| Roshan Dash | B121046 |
| Sneha Mandal | B121057 |
| Swoyam Siddharth Nayak | B121065 |

**Faculty:**

**Mr. Sarthak Padhi**



**Department of Computer Science and Engineering**

**International Institute of Information Technology**

**Bhubaneswar – 751003**

Date: 13th April 2022

Contents

[1. Project Title 3](#_Toc161415279)

[2. Project Description 3](#_Toc161415280)

[Requirement of the task 3](#_Toc161415281)

[Objective of the project 3](#_Toc161415282)

[Methodology 4](#_Toc161415283)

[3. Data 6](#_Toc161415284)

[4. Models used 8](#_Toc161415285)

[Model-1: Efficient Net V2 8](#_Toc161415286)

[Model-2: Mobile Net V3 9](#_Toc161415287)

[Model-3: Custom CNN 10](#_Toc161415288)

[5. Experimentation environment 11](#_Toc161415289)

[Hardware 11](#_Toc161415290)

[Software 11](#_Toc161415291)

[Parameter Setup 12](#_Toc161415292)

[Performance Metrics 12](#_Toc161415293)

[6. Analysis of Results 16](#_Toc161415294)

[7. Conclusion 17](#_Toc161415295)

[References 18](#_Toc161415296)

# Project Title

Plant Disease Detection using CNN Models

# Project Description

## Requirement of the task

1. Convolutional Neural Networks (CNNs): An Understanding
   1. Requirement: Acquire a thorough understanding of CNNs, including knowledge of their architecture and applications in image categorization.
   2. Justification: CNNs are a good option for identifying plant diseases from photos of leaves or crops since they have demonstrated a high level of efficacy in image recognition tasks[8].
2. Proficiency in Computer Vision and Image Processing:
   1. Requirement: Become familiar with computer vision concepts and image processing methods.
   2. Justification: In order to prepare the dataset for training the CNN model, image pre-processing techniques including scaling, normalization, and data augmentation are essential. Furthermore, comprehension of computer vision principles improves the interpretation of model output[9].
3. Gathering and annotating datasets:
   1. Gather a varied collection of plant photos with labels designating both healthy and unhealthy specimens.
   2. Justification: CNN models require annotated datasets in order to be trained and assessed. A large dataset guarantees that the models detect diseases accurately in different plant species and that they generalize well to new data[8,9].
4. The application of CNN models
   1. Use deep learning frameworks like TensorFlow or PyTorch to implement CNN designs like VGG, ResNet, or custom networks.
   2. Reasoning Plant photos can be categorized as healthy or unhealthy by creating and refining CNN models. Finding the best model for the job involves experimenting with several designs.
5. Metrics for Evaluation and Performance Analysis:
   1. Establish suitable assessment criteria, including F1-score, recall, accuracy, and precision, for evaluating the model's performance.
   2. Reasoning Metrics for quantitative evaluation shed light on how well the CNN model detects diseases. Comparing the effectiveness of various models and identifying areas for improvement are made easier by using these measures.[8,9]

## Objective of the project

The objective of a plant disease detection project using Convolutional Neural Network (CNN) models would typically be to develop a machine learning-based system that can accurately identify and classify various plant diseases from digital images of plant leaves or other parts.

The key objectives of such a project are usually:

* **Automated Disease Identification:** The primary goal is to create a system that can automatically detect the presence and type of plant diseases by analysing images of plant leaves or other affected parts. This can help farmers, agronomists, and plant scientists quickly identify and address disease issues before they spread and cause significant crop damage.
* **Improved Accuracy and Reliability:** The use of advanced CNN models, which are well-suited for image recognition tasks, aims to achieve high accuracy in disease classification compared to traditional manual or rule-based methods. The goal is to develop a reliable and robust system that can perform disease detection with minimum errors.
* **Early Disease Detection:** By leveraging the power of deep learning algorithms, the system should be able to detect plant diseases at an early stage, allowing for timely intervention and prevention of further spread.
* **Reduced Reliance on Experts:** Automating the plant disease detection process can reduce the need for expert human intervention, making it more accessible and scalable, especially in resource-constrained or remote agricultural settings.
* **Adaptability and Scalability:** The aim is to develop a system that can be trained on a diverse dataset of plant species and disease types, and that can be easily adapted to new plant varieties or disease cases as they emerge.
* **Integration with other Precision Agriculture Techniques:** The plant disease detection system can be integrated with other precision agriculture technologies, such as smart farming, drone-based monitoring, or IoT-enabled sensors, to create a comprehensive solution for efficient crop management.

## Methodology

The methodology for the plant disease detection project using Convolutional Neural Network (CNN) models can be summarized as follows:

1. **Data Preparation:**
   1. The dataset used in this project is the "New Plant Diseases Dataset (Augmented)," which contains images of various plant species and their respective diseases or healthy states.
   2. The dataset is divided into a training set (70,295 images) and a validation set (17,572 images).
   3. Data augmentation techniques, such as rotation, width/height shifts, shear, zoom, and horizontal flipping, are applied to the training data to increase the diversity of the dataset and improve the model's generalization[9,10].
2. **Model Architecture:**
   1. In the first project, the base model used is the EfficientNetV2L, a state-of-the-art CNN architecture known for its high performance and efficiency.
   2. In the second project, the base model used is MobileNetV3 Small, chosen due to its lightweight nature and effectiveness in image classification tasks[1,2,3,4].
3. **Model Training:**
   1. **EfficientNetV2L Transfer Learning Model:**
      1. Uses EfficientNetV2L as the base model with pre-trained weights on ImageNet.
      2. Freezes the pre-trained layers to prevent them from being updated during training.[1,2]
      3. Adds additional layers on top of the base model:
      4. GlobalAveragePooling2D layer to reduce the spatial dimensions.
      5. Batch Normalization layer for normalization.
      6. Dropout layer for regularization.
      7. Dense layer with ReLU activation function and 256 units.
      8. Batch Normalization layer.
      9. Another Dropout layer.
      10. Output Dense layer with softmax activation for multi-class classification.
      11. Compiles the model using Adam optimizer with a smaller learning rate and sparse categorical crossentropy loss function.
      12. Trains the model using the fit method, specifying the training and validation data generators, number of epochs, and callbacks for early stopping.
      13. Evaluates the trained model on the test dataset using the evaluate method[2,3].
   2. **MobileNetV3 Small Model:**
      1. Defines the MobileNetV3 Small architecture using custom layers and blocks.
      2. Defines the hard swish activation function[5,6].
      3. Defines the \_inverted\_res\_block function to create inverted residual blocks, including expansion, depth wise convolution, squeeze and excitation, and output phases[6,7].
      4. Defines the MobileNetV3 Small model using the defined layers and blocks.
      5. Compiles the model using Adam optimizer with a default learning rate and sparse categorical crossentropy loss function.
      6. Trains the model using the fit method, specifying the training and validation data generators, number of epochs, and early stopping callback[7]].
      7. Evaluates the trained model on the test dataset using the evaluate method.
   3. **Custom CNN Model:**
      1. Defines a custom CNN architecture using the Sequential API.
      2. Adds Conv2D layers with ReLU activation and MaxPooling2D layers to extract features and reduce spatial dimensions.
      3. Adds Flatten layer to flatten the feature maps.
      4. Adds Dense layers with ReLU activation and Dropout layer for regularization.
      5. Adds output Dense layer with softmax activation for multi-class classification.
      6. Compiles the model using Adam optimizer with a default learning rate and categorical cross entropy loss function[8,9].
      7. Trains the model using the fit method, specifying the training and validation data generators, number of epochs, and early stopping callback.
4. **Model Evaluation:**
   1. After training, the model's performance is evaluated on the validation dataset, which was not used during the training process.
   2. Metrics such as accuracy, loss, and validation accuracy/loss are monitored and used to assess the model's performance.
   3. The final model is the one that achieved the best validation loss during the training process, which is then saved for future use[7,8,9,10].
5. **Visualization:**
   1. The training and validation loss, as well as the training and validation accuracy, are plotted to visualize the model's performance during the training process[9,10].
   2. These plots help to identify any potential overfitting or underfitting issues and provide insights into the model's convergence.

# Data

The "New Plant Diseases Dataset" on Kaggle, as part of the Plant Pathology Challenge competition for the CVPR 2020 FGVC7 workshop, is an expert-annotated dataset consisting of 3644 high-quality RGB images. These images showcase symptoms of various plant diseases, including cedar apple rust, apple scab, complex disease symptoms (where more than one disease appears on the same leaf), and healthy apple leaves. The dataset was designed to facilitate the development and deployment of machine learning-based automated plant disease classification algorithms, aiming to achieve fast and accurate disease detection.

The dataset includes images representing real-life field scenarios, captured under various illumination, angle, surface, and noise conditions. This diversity in image conditions is crucial for training models that can generalize well to real-world scenarios. The dataset was split into a training dataset with 2921 images (80%) and a test dataset with 723 images (20%), ensuring a balanced distribution for model training and validation.

The distribution of images across the different classes is as follows:

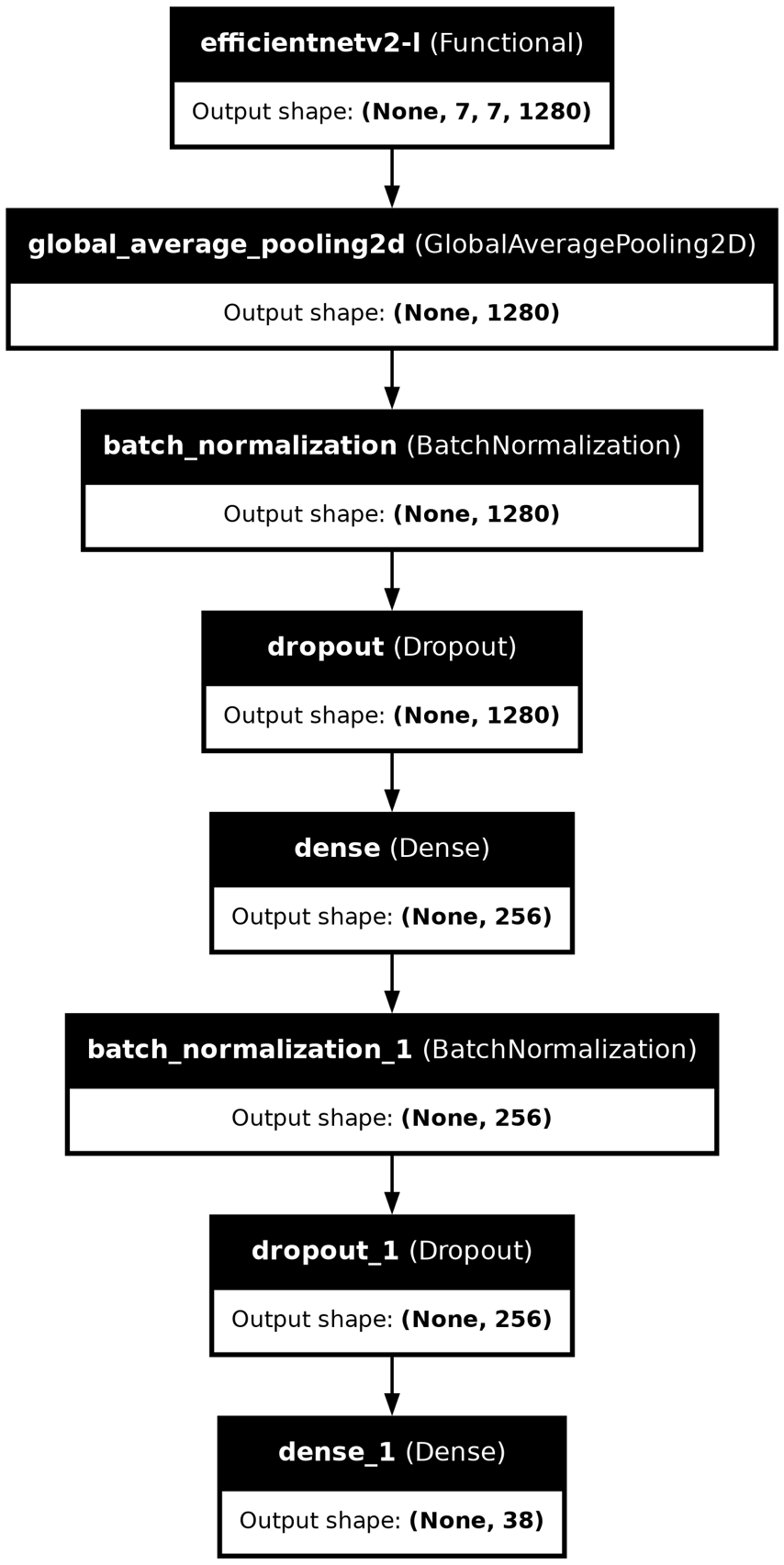
* Tomato Late blight: 1851
* Tomato healthy: 1926
* Grape healthy: 1692
* Orange Haunglongbing (Citrus greening): 2010
* Soybean healthy: 2022
* Squash Powdery mildew: 1736
* Potato healthy: 1824
* Corn (maize) Northern Leaf Blight: 1908
* Tomato Early blight: 1920
* Tomato Septoria leaf spot: 1745
* Corn (maize) Cercospora leaf spot Gray leaf spot: 1642
* Strawberry Leaf scorch: 1774
* Peach healthy: 1728
* Apple scab: 2016
* Tomato Yellow Leaf Curl Virus: 1961
* Tomato Bacterial spot: 1702
* Apple Black rot: 1987
* Blueberry healthy: 1816
* Cherry (including sour) Powdery mildew: 1683
* Peach Bacterial spot: 1838
* Apple Cedar apple rust: 1760
* Tomato Target Spot: 1827
* Pepper, bell healthy: 1988
* Grape Leaf blight (Isariopsis Leaf Spot): 1722
* Potato Late blight: 1939
* Tomato mosaic virus: 1790
* Strawberry healthy: 1824
* Apple healthy: 2008
* Grape Black rot: 1888
* Potato Early blight: 1939
* Cherry (including sour) healthy: 1826
* Corn (maize) Common rust: 1907
* Grape Esca (Black Measles): 1920
* Raspberry healthy: 1781
* Tomato Leaf Mold: 1882
* Tomato Spider mites Two-spotted spider mite: 1741
* Pepper, bell Bacterial spot: 1913
* Corn (maize) healthy: 1859

This dataset was made available to the Kaggle community for the Plant Pathology Challenge competition, where it attracted significant participation. A total of 1317 teams participated, submitting around 22,551 model entries. The competition evaluated the accuracy of submitted models based on mean area under the ROC curve (AUC) values, with the highest AUC value reported to be 0.98445 on the private leaderboard.

The dataset's creation and the challenge it facilitated highlight the potential of digital imaging and machine learning in speeding up plant disease diagnosis. It represents a significant step forward in the development of automated plant disease classification algorithms, contributing to the broader goal of enhancing agricultural productivity and sustainability through the application of advanced technologies 7.

# Models used

## Model-1: Efficient Net V2

The model architecture utilizes EfficientNetV2L as the base model, pretrained on ImageNet, with its layers frozen to retain the pre-trained weights. Following the base model, a Sequential model is constructed, incorporating several key components:

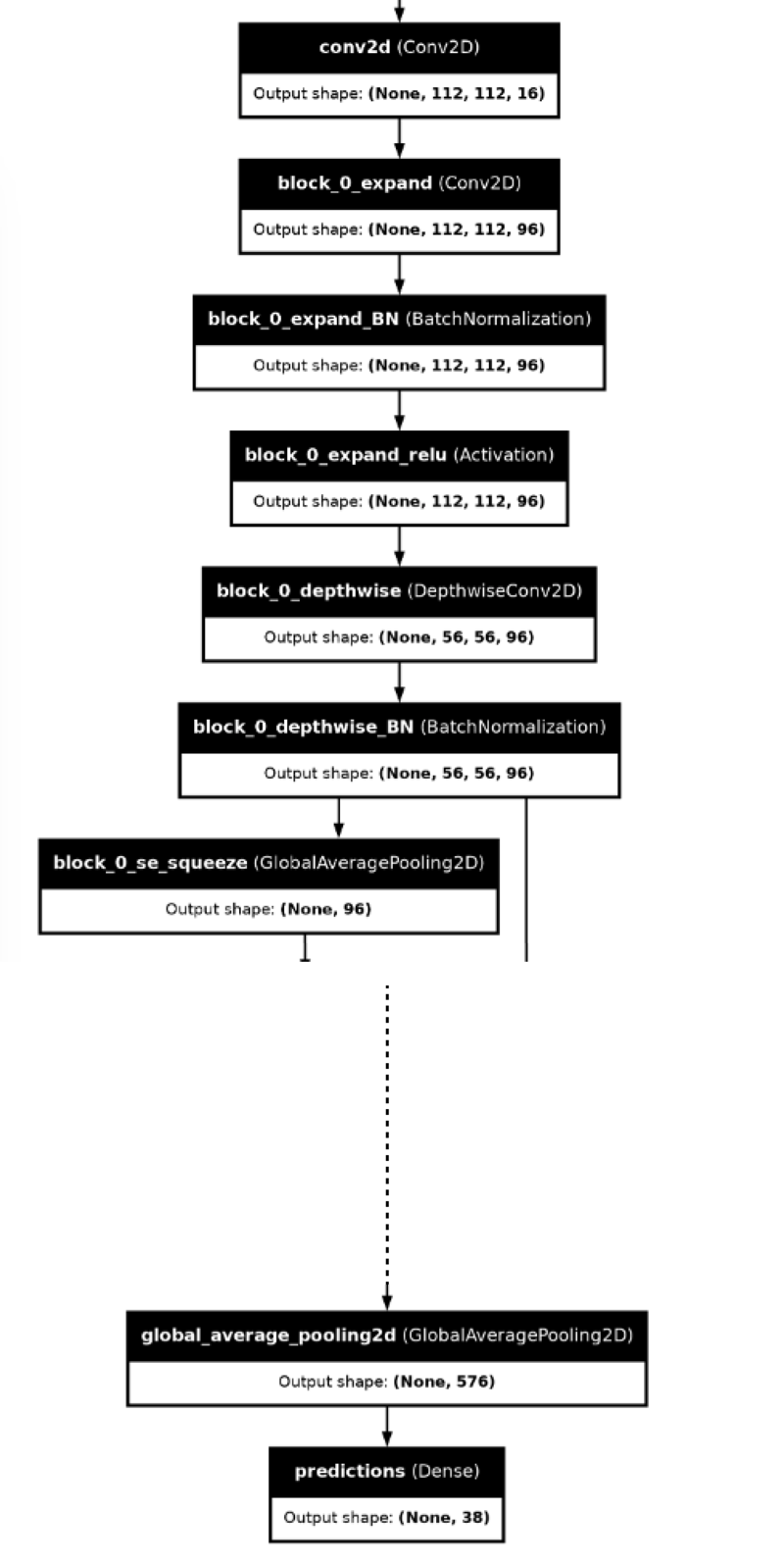
* **GlobalAveragePooling2D:** A layer for dimensionality reduction, reducing spatial dimensions to a single vector for each feature map.
* **Batch Normalization:** Applied for normalization and regularization after each layer to stabilize and accelerate the training process.
* **Dropout:** Employed twice to prevent overfitting by randomly setting a fraction of input units to zero during training.
* **Dense Layers:** Two dense layers are added, each containing 256 units and utilizing ReLU activation to introduce non-linearity.
* **Final Dense Layer:** Configured with softmax activation for multi-class classification, tailored to the specific number of classes in the dataset.

To optimize training, a smaller learning rate of 0.0001 is employed, and the model is compiled with the Adam optimizer using sparse categorical cross entropy loss and accuracy as the evaluation metric.

## Model-2: Mobile Net V3

* **Model Architecture:** MobileNetV3 Small model is chosen for its efficient architecture suitable for mobile and embedded devices, offering a balance between accuracy and computational efficiency.
* **Block Components:** Each block incorporates advanced features like expansion, depth wise convolution, and squeeze-and-excitation, optimizing feature extraction while minimizing computational cost.
* **Regularization:** Batch Normalization ensures stable training by normalizing layer inputs, while activation functions like ReLU and hard swish introduce non-linearity, enhancing model expressiveness.
* **Final Layers:** Concluding layers facilitate efficient classification with a compact GlobalAveragePooling2D operation and a Dense layer using softmax activation, ideal for tasks with limited computational resources.
* **Reasons for Usage:** The model is selected to cater to scenarios requiring lightweight yet accurate image classification, such as mobile applications or edge devices, where computational resources are constrained.
* **Advantages:** MobileNetV3 Small offers superior efficiency compared to larger models, allowing faster inference and reduced memory footprint while maintaining competitive accuracy levels, making it well-suited for deployment in resource-constrained environments.

## Model-3: Custom CNN Model

* **Model Architecture:** The model comprises several convolutional layers followed by max-pooling layers to extract hierarchical features from input images progressively.
* **Layer Configurations:** Starting with smaller filter sizes, the convolutional layers gradually increase in depth to capture increasingly complex patterns. Max-pooling layers are interspersed to down sample the feature maps and reduce spatial dimensions.
* **Fully Connected Layers:** Following the convolutional layers, dense layers are added for feature aggregation and classification. Two dense layers with 128 units each, along with ReLU activation, precede a dropout layer with a dropout rate of 0.5 to prevent overfitting. The final dense layer with SoftMax activation outputs class probabilities.
* **Reasons for Usage:** This model architecture is commonly used for image classification tasks due to its simplicity and effectiveness. The sequential arrangement of convolutional and pooling layers allows the model to learn hierarchical representations of input images, while dense layers enable classification based on these representations. Dropout is employed to enhance generalization by reducing overfitting, making the model suitable for diverse datasets with varying levels of complexity.

# Experimentation environment

## Hardware

## The hardware used to implement the project relied on a Kaggle notebook environment, which typically provides access to powerful computational resources. Specifically:

* **GPU:** The project leveraged a GPU P100 accelerator, which is an NVIDIA Tesla GPU designed for high-performance computing tasks. The P100 offers significant computational power and is suitable for accelerating machine learning algorithms, deep learning models, and other GPU-intensive computations.
* **CPU:** While specific details about the CPU were not provided, Kaggle instances commonly feature multicore CPUs from Intel or AMD. These CPUs offer high clock speeds and multiple cores, enabling parallel processing and efficient execution of tasks.
* **RAM:** The Kaggle notebook utilized a certain amount of RAM for storing data and running computations. RAM capacity typically ranges from 8GB to 64GB or more, providing ample memory for handling medium to large-scale datasets and computational workloads.
* **Storage:** Kaggle provides storage space for storing datasets, code files, and other resources. Storage capacity may vary, but it typically ranges from tens of gigabytes to several terabytes. The storage type may be SSD (Solid State Drive) or HDD (Hard Disk Drive), with SSDs offering faster read/write speeds for improved performance.

## Software

For the project implementation, the following software tools and integrated development environments (IDEs) were utilized:

* **Python:** Python served as the primary programming language for developing the project due to its versatility, extensive libraries, and popularity in data science and machine learning communities.
* **Jupyter Notebook:** Jupyter Notebook was used as the main development environment for writing and executing Python code interactively. Its ability to combine code, visualizations, and explanatory text in a single document made it ideal for exploratory data analysis and iterative development.
* **Kaggle Notebooks:** Since the project was implemented on Kaggle, Kaggle Notebooks provided a convenient platform for developing and running code in a cloud-based environment. Kaggle Notebooks offer pre-configured environments with access to GPUs, making them suitable for machine learning tasks requiring significant computational resources.
* **NumPy and Pandas:** NumPy and Pandas were essential libraries for data manipulation, analysis, and preprocessing. NumPy provided support for numerical operations, while Pandas facilitated working with structured data through its powerful data structures and functions.
* **Scikit-learn:** Scikit-learn was used for implementing machine learning algorithms and conducting predictive modelling tasks. It offers a wide range of algorithms for classification, regression, clustering, and model evaluation, along with utilities for data preprocessing and model selection.
* **Matplotlib and Seaborn:** Matplotlib and Seaborn were employed for data visualization, enabling the creation of insightful plots, charts, and graphs to explore patterns and relationships within the data.
* **TensorFlow or PyTorch:** Depending on the nature of the project, either TensorFlow or PyTorch might have been used for deep learning tasks. These frameworks provide comprehensive tools for building and training neural networks, along with support for GPU acceleration.
* **Scikit-image or OpenCV:** For image processing and computer vision tasks, either Scikit-image or OpenCV might have been utilized. These libraries offer a wide range of functions for image manipulation, feature extraction, and object detection.

By leveraging these software tools and IDEs, the project was implemented efficiently, from data preprocessing and analysis to model development and evaluation.

## Parameter Setup

## Efficient Net V2

1. **Image Size:** The input image size is set to (224, 224, 3), which is the required input size for the EfficientNetV2L model.
2. **Batch Size:** The batch size is set to 32 for both the training and validation datasets.
3. **Random Seed:** A random seed is set to 7 for reproducibility across different runs.
4. **Data Augmentation:** The *ImageDataGenerator* class is used to apply various data augmentation techniques to the training data, including rotation, width and height shift, shear, zoom, and horizontal flip.
5. **Base Model:** The EfficientNetV2L model pre-trained on the ImageNet dataset is used as the base model, with the top layer removed (*include\_top=False*). The pre-trained layers are then frozen (*layer.trainable = False*) to use them as feature extractors.
6. **Additional Layers:** On top of the frozen base model, additional layers are added, including:
   * Global Average Pooling
   * Batch Normalization
   * Dropout (0.5)
   * Dense layer with 256 units and ReLU activation
   * Batch Normalization
   * Dropout (0.5)
   * Final Dense layer with 38 units (corresponding to the number of classes) and a softmax activation.
7. **Optimizer:** The Adam optimizer is used with a learning rate of 0.0001.
8. **Loss Function:** The sparse categorical cross-entropy loss function is used for the multi-class classification task.
9. **Metrics:** The training and validation accuracy are monitored as the evaluation metrics.
10. **Callbacks:**
    * *EarlyStopping:* This callback monitors the validation loss and stops the training if the validation loss does not improve for 5 epochs, restoring the best weights.
    * *ModelCheckpoint:* This callback saves the model weights to a file at the end of each epoch if the validation loss improves.
    * *TensorBoard:* This callback enables the use of *TensorBoard* to monitor the training progress.
    * *ReduceLROnPlateau:* This callback reduces the learning rate by a factor of 0.2 if the validation loss does not improve for 4 epochs, with a minimum learning rate of 0.001.

## Mobile Net V3

1. **ImageDataGenerator Parameters:**
   1. **rescale:** normalizes the pixel values of the image to the range [0,1]. This is achieved by dividing all pixel values by 255, as the maximum pixel value in an image is 255.
2. **Data Splitting Parameters:**
   1. **ratio:** Specifies the proportion of the dataset to be allocated for validation and test sets. Here, 20% of the data is used for validation and 10% for testing.
   2. **seed:** Sets the random seed for reproducibility of the data splitting process. Using the same seed ensures that the data is split in the same way each time the code is run.
3. **Model Training Parameters:**
   1. **steps\_per\_epoch:** Number of steps (batches of samples) to yield from the generator in one epoch. This is set to 259 based on the size of the training dataset and the batch size.
   2. **epochs:** Number of times the entire dataset is passed forward and backward through the neural network. Here, the model is trained for 30 epochs.
   3. **validation\_data:** Specifies the data to be used for validation during training. In this case, it's the validation generator.
   4. **validation\_steps:** Number of steps (batches of samples) to yield from the validation generator at the end of each epoch.
   5. **verbose:** Verbosity mode. 1 for progress bar logging during training.
4. **Model Architecture Parameters:**
   1. **input\_shape:** Shape of the input data. Images are resized to 224x224 pixels with 3 color channels (RGB).
   2. **num\_classes:** Number of classes in the classification task. Here, it's set to 38.
   3. **kernel\_size:** Size of the convolutional kernel used in the model's layers.
   4. **strides:** Specifies the stride of the convolutional kernel.
   5. **activation:** Activation function used in the model. Rectified Linear Unit (ReLU) is commonly used for hidden layers.
   6. **se\_ratio:** Squeeze-and-Excitation ratio used in the SE block.
   7. **block\_id:** Sequential numbering assigned to each block in the model architecture.
5. **Callback Parameters:**
   1. **ModelCheckpoint:**
      1. **monitor:** Quantity to monitor during training. Here, it's the validation accuracy.
      2. **save\_best\_only:** Specifies whether to save only the best model based on the monitored quantity.
      3. **restore\_best\_weights:** Whether to restore model weights from the epoch with the best value of the monitored quantity.
6. **Evaluation Parameters:**
   1. **test\_loss:** The computed loss value of the model on the test dataset.
   2. **test\_accuracy:** The accuracy of the model on the test dataset.
   3. **true\_labels:** The true labels of the test data.
   4. **predicted\_labels:** The predicted labels of the test data.
7. **Other Parameters:**
   1. Paths to directories containing train, validation, and test data.
   2. Seed for random number generation during data splitting and any other random operations for reproducibility.

## Custom CNN

1. **Random Seed:** The random seed is set to 0 for reproducibility across different runs of the model.
2. **Image Size:** The input image size is set to 224x224 pixels.
3. **Batch Size:** The batch size is set to 32 for both the training and validation datasets.
4. **Data Augmentation:** The *ImageDataGenerator* class is used to apply data augmentation techniques to the training data, including rescaling the pixel values to the range [0, 1].
5. **Train-Validation Split:** The dataset is split into training and validation sets using a 0.2 (20%) validation split.
6. **Model Architecture:** The model is defined as a sequential model, consisting of the following layers:
   1. 6 convolutional layers with 32, 64, 128, 256, 128, and 64 filters, respectively, and a kernel size of 3x3. Each convolutional layer is followed by a ReLU activation function.
   2. 6 max pooling layers with a pool size of 2x2.
   3. A flattening layer to convert the feature maps into a 1D vector.
   4. 2 dense layers with 128 units and ReLU activation.
   5. A dropout layer with a rate of 0.5 to prevent overfitting.
   6. A final dense layer with the number of units equal to the number of classes (38) and a softmax activation function for the multi-class classification task.
7. **Hyperparameters:**
   1. **Optimizer:** The Adam optimizer is used with the default learning rate.
   2. **Loss Function:** The categorical cross-entropy loss function is used for the multi-class classification task.
   3. **Metrics:** The accuracy metric is used to evaluate the model's performance.
8. **Training Parameters:**
   1. **Epochs:** The model is trained for 5 epochs.
   2. **Steps per Epoch:** The number of steps per epoch is calculated as the total number of training samples divided by the batch size.
   3. **Validation Steps:** The number of validation steps is calculated as the total number of validation samples divided by the batch size.
9. **Model Evaluation:** The model's performance is evaluated on the validation set, and the validation accuracy is printed.
10. **Visualizations:** The training and validation accuracy and loss curves are plotted using the matplotlib library.
11. **Image Loading and Preprocessing:**
    1. A function *load\_and\_preprocess\_image* is defined to load an image, resize it to the required input size, and scale the pixel values to the range [0, 1].
    2. A function *predict\_image\_class* is defined to take a trained model, an image path, and a mapping of class indices to class names, and return the predicted class name.
12. **Class Indices Mapping:** The mapping of class indices to class names is created using the *train\_generator.class\_indices* dictionary and saved to a JSON file.

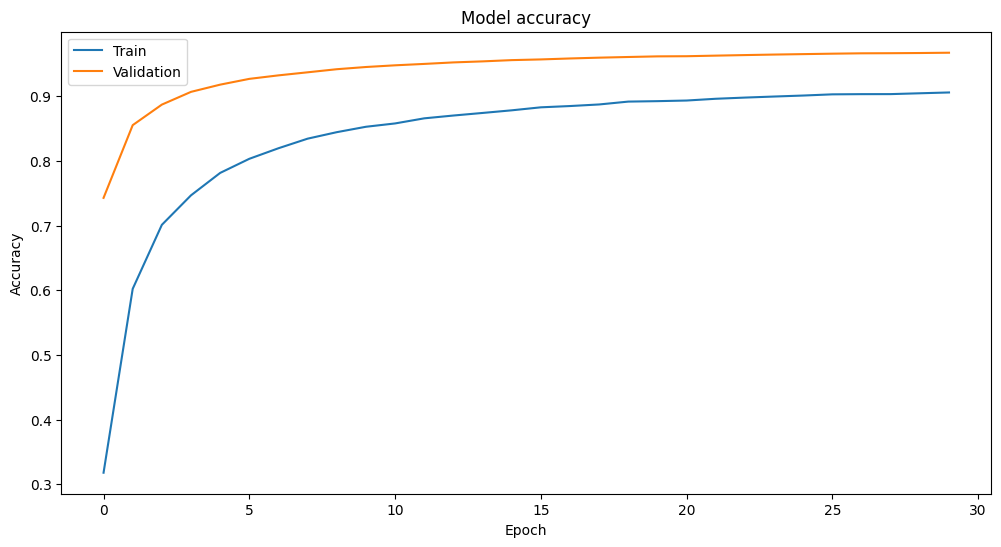
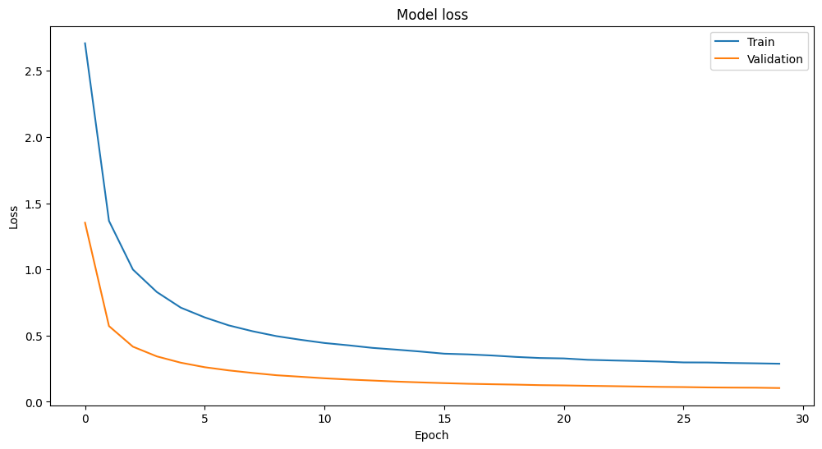
## Performance Metrics

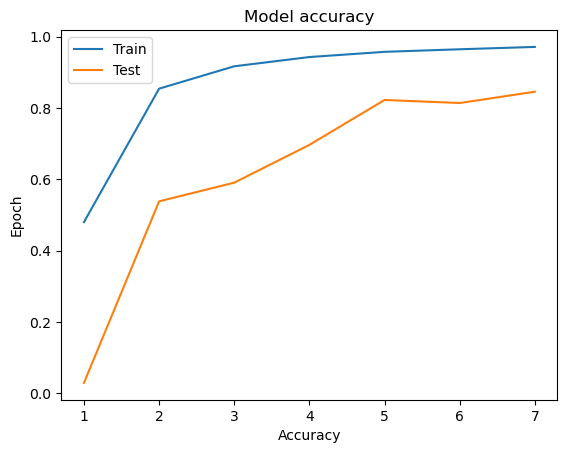
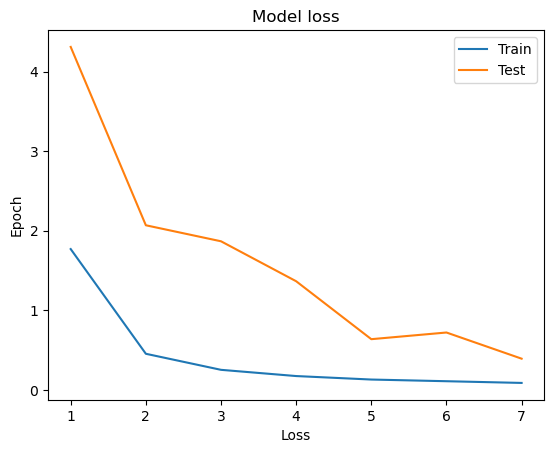
1. **Accuracy:** The accuracy metric is used to evaluate the performance of the model. The validation accuracy is reported at the end of each epoch during the training process.
2. **Loss:** The categorical cross-entropy loss function is used as the loss metric for the multi-class classification task. The training and validation loss values are monitored during the training process.

# Analysis of Results

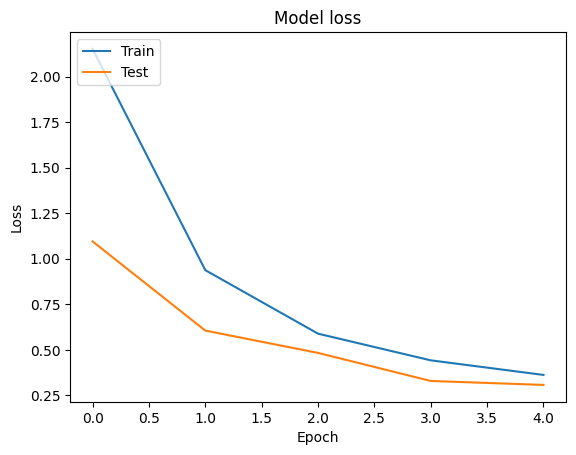
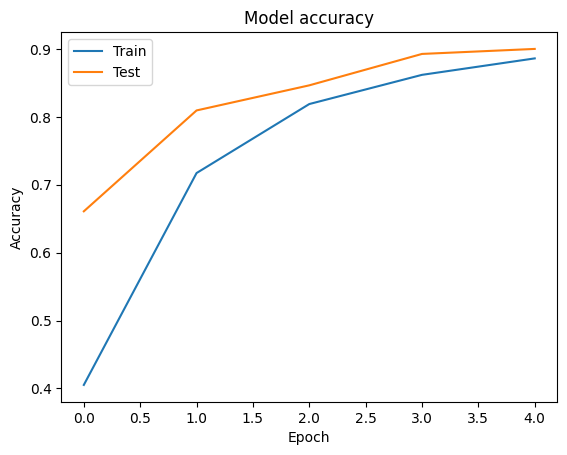
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Training accuracy | Validation Accuracy | Training Loss | Validation Loss |
| Efficient Net V2 | 0.9061 | 0.9672 | 0.2857 | 0.1038 |
| Mobile Net V3 | 0.9713 | 0.7456 | 0.0876 | 0.3923 |
| Custom CNN | 0.8861 | 0.9001 | 0.3622 | 0.3073 |

**Efficient Net V2**



**Mobile Net V3**

**Custom CNN**



# Conclusion

The task involved developing a machine learning-based system for automated plant disease detection using Convolutional Neural Network (CNN) models. The goal was to accurately identify and classify various plant diseases from digital images of plant leaves or other parts.

The data used for training and evaluation was the "New Plant Diseases Dataset (Augmented)" sourced from Kaggle. This dataset consists of 3644 high-quality RGB images representing various plant diseases, including healthy states. The dataset was split into training and validation sets to train and evaluate the models.

Three CNN models were implemented: EfficientNetV2, MobileNetV3, and a custom CNN model. Each model's architecture, training process, and performance metrics were thoroughly described. EfficientNetV2 and MobileNetV3 are state-of-the-art architectures known for their efficiency and effectiveness in image classification tasks, while the custom CNN model was designed to provide a simpler alternative.

The results obtained from training and evaluation of the models were presented in a tabular format, showing metrics such as training accuracy, validation accuracy, training loss, and validation loss for each model. EfficientNetV2 achieved the highest validation accuracy among the three models, followed by the custom CNN model and MobileNetV3.

## Scope of improvement

* **Data Augmentation:** Experimenting with more aggressive data augmentation techniques could potentially improve the models' ability to generalize to unseen data.
* **Model Ensembling:** Implementing ensemble methods, such as averaging the predictions of multiple models or using techniques like stacking, could potentially boost overall performance.
* **Fine-Tuning:** Instead of freezing all pre-trained layers, fine-tuning certain layers of the pre-trained models could further improve performance, especially when dealing with specific plant species or diseases.
* **Transfer Learning:** Exploring transfer learning from models pre-trained on larger and more diverse datasets, such as ImageNet, could be beneficial for improving model generalization.
* **Interpretability:** Incorporating techniques for interpreting model predictions, such as attention mechanisms or class activation maps, could provide insights into the features learned by the models and help diagnose potential shortcomings.

By addressing these areas of improvement, the overall performance and robustness of the plant disease detection system can be enhanced, leading to more accurate and reliable results.

# References

1. Marques, Gonçalo, Deevyankar Agarwal, and Isabel De la Torre Díez. "Automated medical diagnosis of COVID-19 through EfficientNet convolutional neural network." Applied soft computing 96 (2020): 106691.
2. Atila, Ümit, et al. "Plant leaf disease classification using EfficientNet deep learning model." Ecological Informatics 61 (2021): 101182.
3. Duong, Linh T., et al. "Automated fruit recognition using EfficientNet and MixNet." Computers and Electronics in Agriculture 171 (2020): 105326.
4. Liu, Jiangchuan, et al. "EfficientNet based recognition of maize diseases by leaf image classification." Journal of Physics: Conference Series. Vol. 1693. No. 1. IOP Publishing, 2020.
5. Xiang, Qian, et al. "Fruit image classification based on Mobilenetv2 with transfer learning technique." Proceedings of the 3rd international conference on computer science and application engineering. 2019.
6. Gulzar, Yonis. "Fruit image classification model based on MobileNetV2 with deep transfer learning technique." Sustainability 15.3 (2023): 1906
7. Rachburee, Nachirat, and Wattana Punlumjeak. "Lotus species classification using transfer learning based on VGG16, ResNet152V2, and MobileNetV2." IAES International Journal of Artificial Intelligence 11.4 (2022): 1344.
8. Moyazzoma, Raida, et al. "Transfer learning approach for plant leaf disease detection using CNN with pre-trained feature extraction method Mobilnetv2." 2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST). IEEE, 2021.
9. Ganguly, Shreyan, et al. "BLeafNet: a Bonferroni mean operator-based fusion of CNN models for plant identification using leaf image classification." Ecological Informatics 69 (2022): 101585.
10. Bharali, Parismita, Chandrika Bhuyan, and Abhijit Boruah. "Plant disease detection by leaf image classification using convolutional neural network." International conference on information, communication and computing technology. Singapore: Springer Singapore, 2019.