Final Report for the DS 677 - 005 Project
Title: Plant Classification Using Deep Learning

(Harshith Gade(hg355), Sai Nikhil Dunuka(sd2279), Durgaprasad Gudisinti(gd299), Yaswanth Reddy Bushireddy(byr3))

KEYWORDS: Plant Classification, Convolutional Neural Networks (CNN), ResNet50, InceptionV3, Data Augmentation, Early Stopping, Dropout, Overfitting, Regularization (L2), Focused Data Augmentation, Transfer Learning, Hyperparameters, Evaluation Metrics, Classification Report, Confusion Matrix, Precision, Recall, F1-Score, Accuracy, ImageDataGenerator, Multi-Class Classification

ABSTRACT: This project develops an automated plant classification system using deep learning techniques to accurately identify plant species from images, aiding agriculture, conservation, and biodiversity research. Three Convolutional Neural Networks (CNNs) were applied: a custom-built CNN, ResNet50, and InceptionV3. A publicly available plant dataset was pre-processed with normalization, resizing, and data augmentation. Focused data augmentation addressed misclassifications between visually similar species, such as melon and cantaloupe. Evaluation metrics showed ResNet50 and InceptionV3 outperformed the custom CNN, achieving high classification accuracy. This work lays the foundation for deploying deep learning systems in plant monitoring, disease detection, and agricultural research.

Code:https://colab.research.google.com/drive/1GF495Q5ylxeqI8QSIcszSjPuASTwWoG3?usp=sharing
PptSlide:https://docs.google.com/presentation/d/1SmCRU5yVhDOWxZeBxXsk897N45OvmP6o/edit?usp=sharing&ouid=108820247935260726842&rtpof=true&sd=true
Dataset:https://www.kaggle.com/datasets/marquis03/plants-classification/data?select=test

1.INTRODUCTION:

Plant species identification is an essential task for various fields, including agriculture, environmental conservation, and biodiversity monitoring. Traditional methods of plant classification require manual inspection and expert knowledge, which can be slow and prone to errors. As the need for efficient and accurate plant identification grows, automation through deep learning models offers a scalable solution to overcome these challenges.

This project explores the use of deep learning techniques, specifically Convolutional Neural Networks (CNNs), to automate the classification of plant species based on images. CNNs are particularly suited for image classification tasks due to their ability to automatically learn and extract relevant features from images, making them ideal for handling the complexity of plant recognition.

The project investigates three different models: a custom CNN model designed from scratch, ResNet50, and InceptionV3. These models are trained and fine-tuned to classify images of various plant species. The goal is not only to achieve high classification accuracy but also to understand the performance differences between these models when applied to the task of plant species identification.

Furthermore, data augmentation techniques are applied to improve the models' generalization abilities, especially for challenging classes with similar visual features. This project aims to contribute to the growing body of research in the field of computer vision for ecological and agricultural applications, offering potential for real-time plant identification and monitoring.

2. ABOUT DATASET:

The dataset used for this project is a publicly available plant classification dataset, which contains images of 30 different plant species. Each species has a collection of images that are used for training, validation, and testing the models. The dataset is organized into subfolders, where each subfolder corresponds to a specific plant species, containing several images of that species.

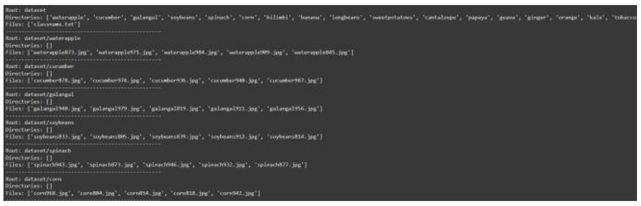
The dataset includes a total of 6000 images, with each species having an equal number of images (No Class Imbalance). The images have been labeled accordingly, ensuring that each image is associated with its correct class.

The dataset structure is as follows:

Root Folder (Dataset Path):

30 subfolders, each named after a plant species.

Each subfolder contains images of the respective plant species.



```
Root: dataset
Directories: ['waterapple', 'cucumber', 'galangal', 'soybeans', 'spinach', 'corn', 'bilimbi', 'banana', 'longbeans', 'sweetpotatoes', 'cantaloupe', 'papaya', 'guava', 'ginger', 'orange', 'kale', 'tobacco
Files: ['classname.txt']
--------------------------------------------------
Root: dataset/waterapple
Directories: []
Files: ['waterapple873.jpg', 'waterapple971.jpg', 'waterapple984.jpg', 'waterapple909.jpg', 'waterapple845.jpg']
--------------------------------------------------
Root: dataset/cucumber
Directories: []
Files: ['cucumber878.jpg', 'cucumber974.jpg', 'cucumber936.jpg', 'cucumber940.jpg', 'cucumber987.jpg']
--------------------------------------------------
Root: dataset/galangal
Directories: []
Files: ['galangal940.jpg', 'galangal979.jpg', 'galangal819.jpg', 'galangal911.jpg', 'galangal956.jpg']
--------------------------------------------------
Root: dataset/soybeans
Directories: []
Files: ['soybeans833.jpg', 'soybeans805.jpg', 'soybeans839.jpg', 'soybeans912.jpg', 'soybeans814.jpg']
--------------------------------------------------
Root: dataset/spinach
Directories: []
Files: ['spinach943.jpg', 'spinach873.jpg', 'spinach946.jpg', 'spinach932.jpg', 'spinach877.jpg']
--------------------------------------------------
Root: dataset/corn
Directories: []
Files: ['corn968.jpg', 'corn884.jpg', 'corn854.jpg', 'corn818.jpg', 'corn942.jpg']
--------------------------------------------------
```

3.DATA PREPROCESSING:

Preprocessing is a critical step in any machine learning project, particularly in image-based tasks. The images in the dataset are of varying sizes, resolutions, and formats, which necessitate standardization before they can be fed into the deep learning models. Below are the key preprocessing steps performed:

Resizing:

All images were resized to a uniform size of 224x224 pixels to match the input dimensions required by the models (ResNet50, InceptionV3). Resizing ensures that the models receive consistent input images.

Normalization:

The pixel values of the images were scaled to a range between 0 and 1 by dividing the pixel values by 255. This helps in faster convergence during training, as it standardizes the input range.

Data Augmentation:

Data Augmentation techniques were applied to artificially increase the diversity of the dataset and reduce overfitting. These augmentations include:

Rotation, width and height shift, shear transformations, zooming, and horizontal flipping.

Specific augmentations, such as focused augmentation, were applied to address class imbalance and misclassifications between visually similar classes.

Data Splitting:

The dataset was split into training, validation, and test sets. The training set consisted of 70% of the total images, the validation set was 15%, and the test set was 15%. This split ensured that the models were trained on one subset of the data and validated and tested on unseen data to evaluate generalization.

By performing these preprocessing steps, the data is transformed into a form that is ready for model training, enabling the models to learn from a standardized and diverse set of images. The augmentation and data splitting strategies further ensure that the models are capable of handling unseen data and can generalize well.

4.CLASS DISTRIBUTION AND PLANT CLASSES:

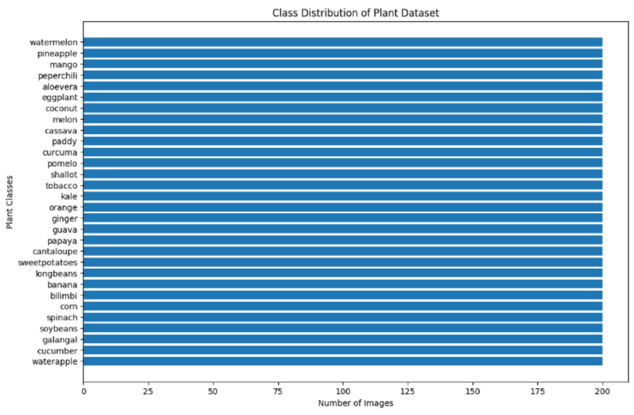
To understand the balance of classes in our plant classification dataset, we first analyzed the distribution of images across different plant species. Since the dataset contains images categorized into 30 plant classes, it was crucial to check if the classes were evenly distributed, as class imbalance can lead to biased model performance.

We utilized the following steps to analyze the dataset:

Identifying Plant Classes: We used the os module to read the directory structure of the dataset, extracting the names of plant classes (i.e., subdirectories representing different plant species). These class names were stored in a list, allowing us to easily track the species present in the dataset.

Counting Images per Class: For each plant class, we counted the number of images using os.listdir(), which gives the number of files (images) in each directory. This helped us understand how many images we had for each class and whether the classes were evenly distributed.

Class Distribution Visualization: To visually assess the class distribution, we generated a bar plot using matplotlib. This plot displayed the number of images per class, giving a clear view of whether some classes were underrepresented or overrepresented.



Model Development: Custom CNN

The first step in our plant classification project was to create a custom Convolutional Neural Network (CNN) from scratch. This was done to establish a baseline model before experimenting with more advanced pre-trained models like ResNet50 and InceptionV3.

The custom CNN model was selected:

Simplicity and Control: Building a CNN from scratch gave us complete control over the architecture, allowing us to experiment with different configurations and layers, such as the number of filters, kernel sizes, and fully connected layers.

Understanding the Fundamentals: It allowed us to gain a deeper understanding of the core principles of deep learning models and CNNs, including feature extraction through convolutional layers, pooling layers for dimensionality reduction, and the use of dense layers for final classification.

Baseline Performance: By starting with a custom CNN, we could benchmark its performance against more sophisticated architectures, providing us with a clear measure of how well more advanced models like ResNet50 or InceptionV3 could perform.

Model Architecture:

We built a simple CNN architecture that consists of multiple convolutional and pooling layers followed by fully connected layers. This architecture is capable of learning hierarchical features, such as edges, textures, and object parts in the images.

The architecture consists of the following layers:

Convolutional Layers: These layers apply filters (kernels) to input images and detect important features like edges and shapes. We used 3x3 filters with ReLU activation to add non-linearity and increase the network's learning capacity.

Layer 1:

Number of filters: 32

Filter size: 3x3

Activation function: ReLU

Layer 2:

Number of filters: 64

Filter size: 3x3

Activation function: ReLU

Layer 3:

Number of filters: 128

Filter size: 3x3

Activation function: ReLU

Max-Pooling Layers: These layers reduce the spatial dimensions of the feature maps, effectively lowering the computational cost while preserving the most important information.

Pooling size: 2x2

Fully Connected Layers:

These layers are used to make the final classification decisions. The output layer uses a soft-max activation function, which provides class probabilities, as this is a multi-class classification task (with 30 plant species).

Dense Layer 1:

Number of neurons: 128

Activation function: ReLU

Output Layer:

Number of neurons: 30 (one for each class)

Activation function: Softmax

Softmax is used to calculate the probabilities of each class, ensuring that the sum of the output values is 1.

Dropout Layer: We incorporated a Dropout layer to prevent overfitting during training by randomly disabling some neurons during each training step. This helps improve the generalization ability of the model.

Dropout Layer:

Dropout rate: 0.5

The model was compiled with the following:

Adam Optimizer: We used the Adam optimizer due to its ability to adapt the learning rate during training, which helps improve convergence and stability.

Adam Optimizer with a learning rate of 0.0001

Categorical Cross entropy Loss: Since our task is a multi-class classification problem (30 plant species), we used categorical cross entropy as the loss function. This function is designed to minimize the error between the predicted and actual class probabilities.

Accuracy Metric: The model was trained to optimize for accuracy, which helps us track how well the model is classifying images correctly.

Performance Evaluation: After training, the model's performance was evaluated using several metrics:

Training and Validation Accuracy: To track the improvement of the model over time.

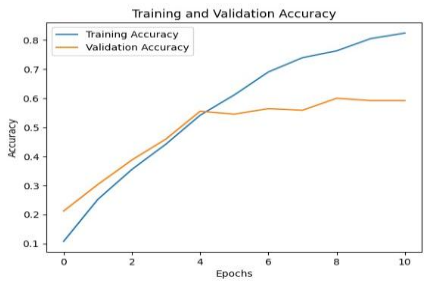
Confusion Matrix: To check which classes were misclassified and provide insights into areas of improvement.

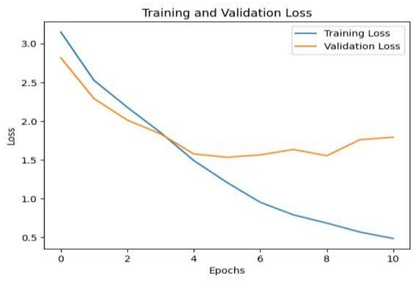
Precision, Recall, F1-Score: These metrics were used to assess the model's performance more comprehensively, especially for classes with a higher number of misclassifications.

5.TRAINING & VALIDATION:

The model was trained using data augmentation techniques such as rotation, flipping, and zooming to improve its ability to generalize and reduce overfitting. Additionally, we used a training-validation split to ensure that the model's performance was evaluated on unseen data throughout the training process. We trained the model for a certain number of epochs (10) and monitored the accuracy and loss during both training and validation.

In our plant classification project, overfitting was identified as a key issue after training the model for several epochs. To address this, we implemented the following strategies to improve the model's generalization ability and prevent overfitting.





Techniques Used to Prevent Overfitting:

1. L2 Regularization:

What is it? L2 regularization, also known as weight decay, adds a penalty to the loss function based on the square of the magnitude of the model weights. This helps prevent the model from fitting too closely to the training data and reduces the risk of overfitting.

Why we used it?

We applied L2 regularization to the convolutional and fully connected layers. This technique discourages the network from learning excessively large weights, which could lead to a model that is too sensitive to small fluctuations in the training data. By penalizing large weights, L2 regularization helps the model focus on more relevant features that generalize better to unseen data.

How it is implemented?

The L2 regularization term was added to the loss function by including it in the layer definition. This penalizes large weight values during the backpropagation process.

2. Increased Dropout:

What is it?

Dropout is a regularization technique where randomly selected neurons are "dropped" or set to zero during training. This prevents the model from becoming overly reliant on any single neuron, helping to generalize better to new data.

Why we used it?

We kept the dropout rate at 0.5 in the fully connected layer to aggressively regularize the model and prevent overfitting. Dropout is particularly useful when training deep neural networks, as it forces the network to learn redundant representations, ensuring robustness when applied to unseen data.

How it was implemented?

The dropout layer was placed after the fully connected layer, with a dropout rate of 50%. This means that during training, half of the neurons in this layer are randomly turned off to reduce overfitting.

3. Early Stopping:

What is it?

Early stopping is a technique used to halt the training process when the validation loss stops improving, preventing the model from overfitting to the training data.

Why we used it?

After several epochs, if the model continued training without improvement in validation performance, it would have started to memorize the training data rather than generalize well. Early stopping monitors the validation loss and stops training once the loss stops improving, saving time and preventing overfitting.

How it was implemented?

The training process was monitored for validation loss. We set a patience value of 5 epochs, meaning training would stop if the validation loss did not improve for 5 consecutive epochs.

4. Data Augmentation:

What is it?

Data augmentation artificially increases the size of the training dataset by applying random transformations (such as rotation, zoom, flipping) to the existing images. This helps the model generalize better by providing it with more diverse training examples.

Why we used it?

We applied data augmentation to ensure that the model did not memorize the training data and to expose it to various image variations, such as rotated or flipped versions of the same plant species. This technique helps the model become more robust and handle new, unseen images more effectively.

How it was implemented?

The ImageDataGenerator class in Keras was used to apply random transformations to the images during training. This included rotation, zoom, width and height shifts, horizontal flipping, and more.

By using these techniques, we were able to reduce overfitting and improve the model's ability to generalize. The custom CNN, which initially suffered from overfitting, was able to perform better on unseen data, leading to improved accuracy and loss metrics on the validation and test sets.

6.RESULTS:

After applying various regularization techniques such as L2 regularization, dropout, early stopping, and data augmentation, we trained our custom CNN model and evaluated its performance on the test set. Here are the key results:

Test Accuracy and Loss:

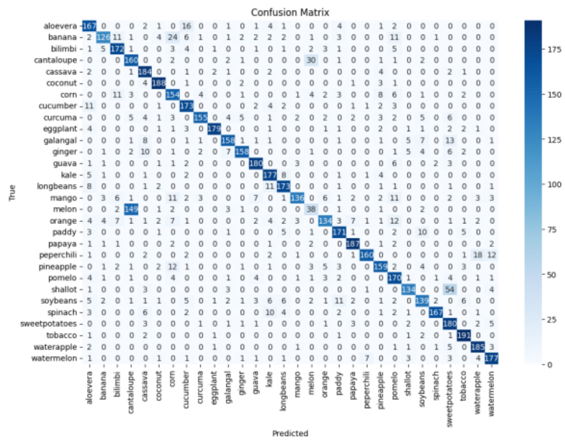
Accuracy: 80.53%

Loss: 0.7339

These results show that the model achieved a good accuracy of over 80% on the unseen test data, suggesting it learned the relevant features during training and generalized well to new images. The test loss indicates how well the model's predictions match the true labels, with a value of 0.7339 showing that it is performing reasonably well.

7.VISUALIZATION AND METRICS:

Confusion Matrix: The confusion matrix visualized the predicted vs. actual labels, highlighting which classes were misclassified. Classes with low precision and recall, such as melon, were identified.



Key Observations:

The model performed descent overall, but there were certain plant classes that were misclassified (mainly melon and cantaloupe). The results suggest that the melons were particularly difficult to distinguish from cantaloupe due to their similar appearance.

Focused Data Augmentation for Misclassified Classes (Melon and Cantaloupe)

In our initial model, we observed a significant misclassification issue between the melon and cantaloupe classes. Despite the overall good performance of the model, these two classes were often confused, which led to suboptimal accuracy for both. This misclassification was particularly evident in the Confusion Matrix, where the melon class was often predicted as cantaloupe, and vice versa.

To address this issue and enhance the model's ability to correctly classify these two similar-looking plant species, we implemented Focused Data Augmentation. The objective of this technique was to:

Increase diversity in the training data for the underperforming classes (melon and cantaloupe).

Improve the model’s generalization by generating more varied examples for these two classes, allowing the model to learn more robust and distinguishing features.

What We Did:

Focused Data Augmentation involves applying specific augmentation techniques to the images from the problematic classes. For melon and cantaloupe, we applied different sets of augmentation transformations to artificially expand the diversity of the training data. This approach allowed the model to encounter various variations of these classes and learn better representations.

Focused Augmentation for Melon:

We applied the following transformations to the melon class:

Rotation: Randomly rotate the images up to 30 degrees to simulate different orientations of the fruit.

Shift: Apply random width and height shifts up to 20% to simulate slight translations of the object.

Shear: Apply random shear transformations up to 20% to simulate perspective changes.

Zoom: Zoom in and out by up to 20% to simulate different levels of proximity to the object.

Horizontal Flip: Flip the images horizontally to introduce symmetry.

Fill Mode: Use nearest fill mode to fill missing pixels during transformations.

These transformations were applied to increase diversity within the melon class and allow the model to learn more generalized features.

Focused Augmentation for Cantaloupe:

Similar transformations were applied to the cantaloupe class:

Rotation Range: Rotate the images randomly up to 50 degrees to introduce variations in angle.

Width and Height Shifts: Apply random shifts up to 30%.

Shear and Zoom: Introduce shear and zoom ranges to simulate different perspectives and proximities.

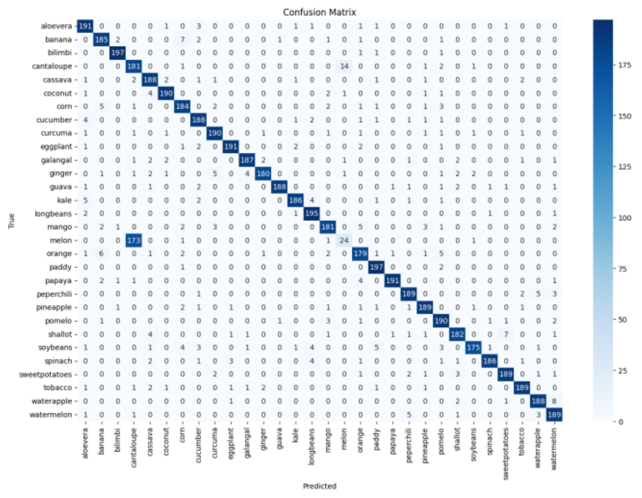
Brightness and Channel Shifts: Adjust the brightness by a range of 0.8 to 1.2 to simulate lighting changes. Additionally, channel shifts were used to adjust the color channels, making the images more robust to lighting and environmental changes.

Why Focused Data Augmentation?

Increase Class Diversity: By applying data augmentation specifically to the melon and cantaloupe classes, we increased the number of variations the model saw during training. This made the model more robust and able to distinguish between these similar-looking classes more accurately.

Enhance Generalization: Focused augmentation helps in reducing overfitting by forcing the model to learn generalized features from a more diverse set of images. This allows the model to better handle new, unseen images from these classes.

Address Imbalance: Though the classes were balanced in terms of the number of samples, these classes were inherently harder to classify due to their visual similarity. Augmenting the data for these specific classes provided them with more varied examples, which improved their classification performance.



8.LIMITATION AND CHALLENGES:

Despite the use of Focused Data Augmentation to address misclassification between melon and cantaloupe, the model still struggled due to several key factors:

Insufficient Variations in the Training Dataset:

Problem: Both fruits share similar visual features (shape, color, texture), and the dataset lacked enough diverse examples.

Impact: The model struggled to generalize and distinguish between them due to the absence of real-world variations (e.g., different lighting, ripeness).

Similar Visual Features:

Problem: Melons and cantaloupes have intrinsically similar characteristics, making them hard to differentiate.

Impact: Even with data augmentation, the model failed to learn strong distinguishing features, leading to misclassification.

Model's Limitations:

Problem: Despite using regularization techniques, the model showed lower performance on these two classes.

Impact: This suggests the model's architecture might need further refinement to handle such similar classes effectively.

Data Representation Issues:

Problem: The dataset didn't provide sufficient variety (e.g., different cultivars, ripeness stages).

Impact: Limited variation in the data hindered the model's ability to make accurate predictions for these classes.

9.RESNET50:PRE-TRAINED MODEL

What is ResNet50?

ResNet50 is a deep convolutional neural network architecture that is part of the Residual Networks (ResNet) family, which was developed by Microsoft Research. The key feature of ResNet is the use of residual connections (skip connections) that help combat the vanishing gradient problem and enable the training of very deep networks with hundreds or thousands of layers.

Residual Connections: These connections allow the network to bypass certain layers, effectively enabling the model to learn identity mappings. This makes it easier for the network to learn, especially in very deep networks.

50 Layers: ResNet50 has 50 layers, which provides a good balance between model depth and efficiency for complex tasks such as image classification.

Why We Used ResNet50:

Pre-trained Weights: ResNet50 is available with pre-trained weights, trained on the ImageNet dataset, which contains millions of labeled images. Using these pre-trained weights as a starting point allows our model to benefit from transfer learning, where the model leverages previously learned features and can be fine-tuned on our specific plant classification dataset.

Improved Performance: Pre-trained models like ResNet50 are well-optimized for image classification tasks. Using a pre-trained model helped us achieve better performance compared to training a model from scratch. ResNet50 is especially effective for tasks where the classes are visually complex, as it can learn fine-grained features without needing to start the learning process from scratch.

Handling Deeper Networks: Given that ResNet50 is a deep model with 50 layers, it is capable of learning complex features from the images, which is essential when distinguishing between plant species with subtle differences in their visual appearance.

Working of ResNet50 in Our Project:

Feature Extraction: ResNet50 acts as a feature extractor, where the early layers capture low-level features (such as edges and textures), and deeper layers capture high-level features (such as shapes and structures). These learned features are then passed to the fully connected layers to make predictions.

Transfer Learning: Instead of training the model from scratch, we used ResNet50 with pre-trained weights. This allowed us to fine-tune the network to our specific dataset (plant classification). The pre-trained weights helped the model quickly learn useful features, which were adapted to classify plant species.

Fine-tuning: We froze the pre-trained layers of ResNet50 (i.e., set their weights as non-trainable) initially and only trained the newly added layers (dense layers). This helped in adapting the model to our plant classification problem without overfitting, as the pre-trained layers already learned general features.

Final Layers: After the base ResNet50 model, we added a global average pooling layer to reduce the dimensionality, followed by a fully connected layer and the output layer with 30 units (one for each class) and a softmax activation function for multi-class classification.

Results Compared to Custom CNN:

Accuracy: ResNet50 consistently outperformed the Custom CNN in terms of test accuracy, precision, and recall. The pre-trained ResNet50 model's accuracy was around 92%, while the custom CNN model achieved around 81%.

Faster Convergence: The ResNet50 model converged much faster due to the use of pre-trained weights, whereas the custom CNN required more epochs to reach optimal performance. This is because the pre-trained ResNet50 model already had learned useful features on ImageNet.

Better Generalization: The ResNet50 model exhibited better generalization to unseen data compared to the custom CNN, likely due to its deeper architecture and pre-training on a large and diverse dataset like ImageNet.

Hyperparameters and Optimizers Used for ResNet50:

Learning Rate: We used a learning rate of 0.0001 with the Adam optimizer. This low learning rate was chosen to avoid large weight updates during fine-tuning, ensuring that the pre-trained weights were not drastically altered.

Adam Optimizer: The Adam optimizer was chosen for its adaptive learning rate capabilities. It helps the model converge faster and more efficiently, especially when fine-tuning a pre-trained model.

Loss Function: We used categorical cross-entropy as the loss function because this is a multi-class classification problem (with 30 plant species to classify).

Metrics: We monitored accuracy during training to evaluate the model’s performance at each epoch.

Training Strategy:

Initially, we froze the weights of the pre-trained ResNet50 layers to focus on training the custom layers we added (global average pooling, fully connected layer, and output layer).

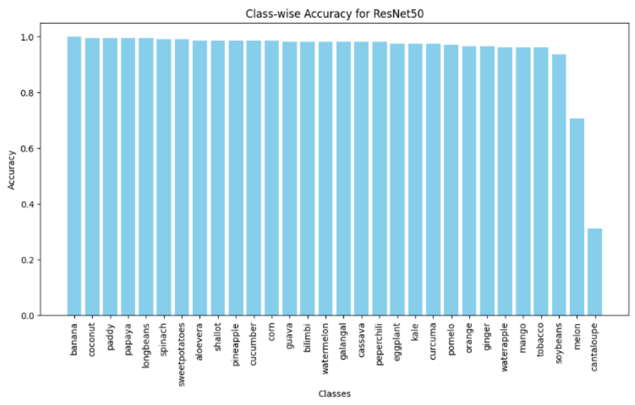
After several epochs, we un-freezed the ResNet50 layers and fine-tuned the entire network at a lower learning rate (0.00001). This allows the model to learn more specific features from the plant images without distorting the useful features learned during the pre-training.

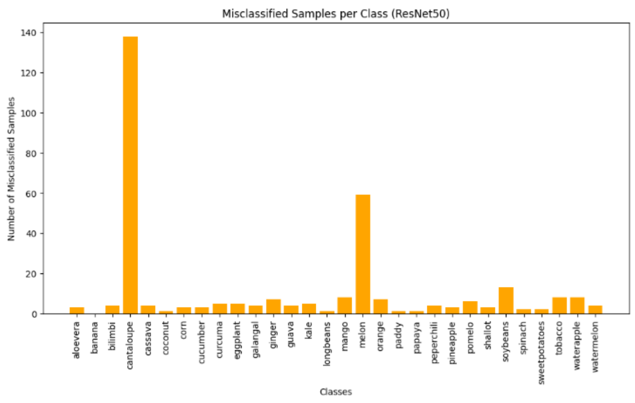
Limitations and Results:

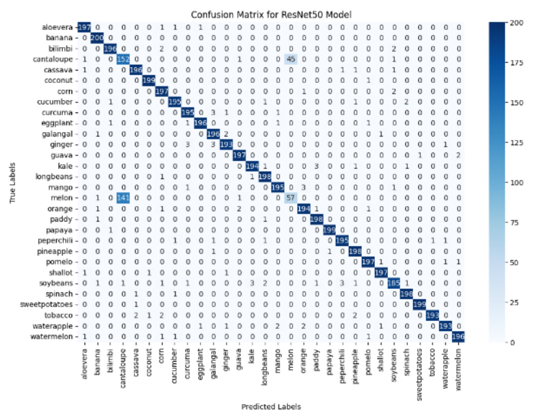
Even though ResNet50 provided better performance than the custom CNN, it still misclassified the melon and cantaloupe classes due to Insufficient Data Variations in the DataSet.

Although ResNet50 benefits from pre-trained weights, the model's ability to generalize is still dependent on the quality and diversity of the training data. As the training data doesn't capture enough variations in appearance (e.g., different ripeness stages, lighting conditions, or angles), the model didn’t perform well on these classes. Our dataset likely lacked such diversity for melon and cantaloupe.

While fine-tuning the ResNet50 model improved its performance, it still couldn't fully discriminate between these two similar classes. The fine-tuning process helped adapt the model to our plant dataset, but it didn’t solve the fundamental issue of their visual similarity.







Despite ResNet50's strong performance in most cases, the melon and cantaloupe classes remained problematic due to their visual similarity and the lack of sufficient diversity in the dataset. More targeted efforts like better data collection, further augmentation, and even exploring advanced models or techniques like domain adaptation could help improve the model’s ability to classify these challenging classes.

10.INCEPTIONV3: PRE-TRAINED MODEL

What is InceptionV3?

InceptionV3 is a deep convolutional neural network architecture that was designed by Google for large-scale image recognition tasks. It is a part of the Inception series of models, known for its efficiency and performance in computer vision tasks.

Multiple Filter Sizes: InceptionV3 employs multiple filter sizes (1x1, 3x3, 5x5 convolutions) in parallel to capture a wide range of features at different scales. This is known as the Inception module.

Factorization: It uses factorization techniques like 1x1 convolutions for dimensionality reduction and 3x3 convolutions to maintain feature representation while reducing computational complexity.

Auxiliary Classifiers: InceptionV3 includes auxiliary classifiers during training to encourage better feature extraction. These auxiliary classifiers provide additional gradients, helping the model converge faster.

Global Average Pooling: Instead of using fully connected layers, InceptionV3 uses global average pooling, which reduces the number of parameters and helps reduce overfitting. This also improves the model's generalization ability.

Why We Used InceptionV3:

Pre-trained Weights: Like ResNet50, InceptionV3 is available with pre-trained weights on ImageNet, which is a large dataset with millions of labeled images. Using these weights as a starting point allows us to benefit from transfer learning, where the model leverages the features it learned from ImageNet and adapts them to our plant classification task.

Multi-scale Feature Extraction: InceptionV3’s ability to capture features at multiple scales due to its parallel convolution layers makes it particularly effective at distinguishing between complex objects. This is useful when dealing with classes that have subtle visual differences, such as the different plant species.

Efficiency: InceptionV3 is computationally more efficient than deeper models like ResNet and offers a good balance between model complexity and performance, making it suitable for our task of classifying 30 plant species.

Better Performance on Complex Visual Features: The architecture's ability to capture complex, multi-scale features helped improve accuracy, especially for classes with more subtle visual differences.

Working of InceptionV3 in Our Project:

Feature Extraction: InceptionV3, like ResNet50, serves as a feature extractor, learning the underlying features of the plant images through its pre-trained layers. These learned features (such as edges, textures, and shapes) are then used by the fully connected layers to predict the class of each image.

Transfer Learning: We used InceptionV3 with its pre-trained weights, allowing the model to start with a solid foundation of learned features from ImageNet. The pre-trained layers were frozen initially, and only the new layers (such as the fully connected layers) were trained on our plant classification dataset.

Fine-tuning: After training the custom layers, we unfreezed the InceptionV3 layers and fine-tuned the entire model. This enabled the model to adjust to our plant dataset while retaining the useful features learned during pre-training.

Final Layers: After the base model (InceptionV3), we added a global average pooling layer, followed by a fully connected layer and a softmax output layer to classify the 30 plant species.

Hyperparameters and Optimizers Used for InceptionV3:

Learning Rate: We used a learning rate of 0.0001 with the Adam optimizer. This learning rate was set to allow the model to adjust weights gradually without overfitting or oscillating.

Adam Optimizer: We chose Adam for its adaptive learning rate capabilities. Adam is well-suited for training deep models like InceptionV3, as it helps in faster convergence and avoids overshooting during gradient updates.

Loss Function: The loss function used was categorical cross-entropy, as the task is a multi-class classification problem (30 plant species to predict).

Metrics: We tracked accuracy during training to monitor how well the model was performing with each epoch.

Results of InceptionV3 Compared to Custom CNN and ResNet50:

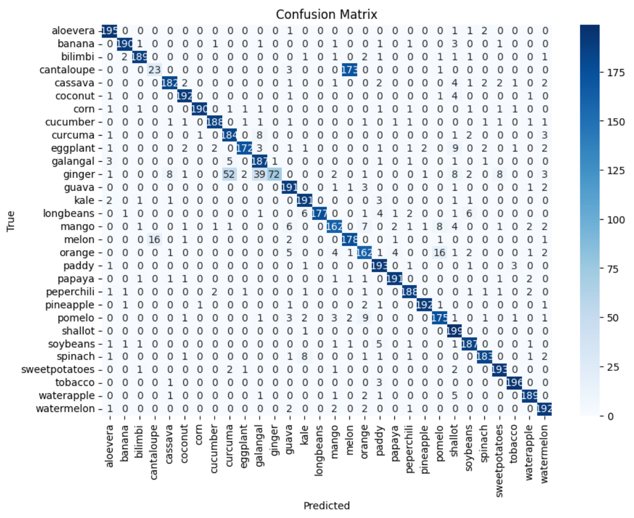
Accuracy: InceptionV3 achieved a Final Training Accuracy of 87.50%, which is an improvement compared to Custom CNN (81%) and ResNet50 (95%).

InceptionV3 performed slightly worse than ResNet50 but outperformed the Custom CNN in terms of training accuracy and generalization ability.

Loss: The Final Training Loss of InceptionV3 was 0.4061, indicating better model performance in terms of minimizing errors compared to Custom CNN and ResNet50.

Improved Performance on Complex Classes: InceptionV3's ability to extract multi-scale features contributed to its better performance in distinguishing between more complex plant species, providing better accuracy for a majority of the plant classes.

Training Speed: InceptionV3 was slightly slower in training compared to ResNet50, but it still performed descent within the required time frame.



11.FUTURE WORK:

Expansion of Dataset: The model can perform better with a larger, more diverse dataset. More examples for underrepresented classes will improve the model's generalization ability.

Model Exploration: We would like to explore Vision Transformers (ViT) or DenseNet for potentially better performance in terms of feature extraction and classification accuracy.

Deployment: Deploying the model into a practical application, like a mobile or web app, would make plant classification accessible in real-time.

12.CONCLUSION:

This section is to wrap up the insights gained from the project, including the challenges faced, the effectiveness of data augmentation, and the comparison between models.

By automating plant classification, we can improve the efficiency of plant monitoring, disease detection, and biodiversity studies, making the process scalable and easily accessible.

13.REFERENCES:

[1]https://ieeexplore.ieee.org/document/10568579?utm_source

[2]https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2023.1286088/full?utm_

[3]https://link.springer.com/chapter/10.1007/978-981-97-1320-2_10?utm_

[4]https://www.frontiersin.org/journals/physiology/articles/10.3389/fphys.2023.1126780/full?utm_

[5]https://arxiv.org/pdf/1706.03736

[6]https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/?utm_

[7]https://ieeexplore.ieee.org/abstract/document/9399342?utm_