# Evaluating ANN vs CNN for Real-World Medicinal Plant Identification Using Deep Learning Frameworks

*Dr. T. Swathi Priyadarshini,*
*Assistant Professor,*
*Neil Gogte Institute of Technology,*
*Hyderabad, India*

*Kumbala Deekshitha*
*Neil Gogte Institute ofTechnology,*
*Hyderabad, India*
*kumbaladeekshitha2@gmail.com*

*Madhu Tanmayee Dhataram*
*Neil Gogte Institute ofTechnology,*
*Hyderabad, India*
*madhutanmayee.dhataram@gmail.com*

*Velukanti Jasnavi*
*Neil Gogte Institute ofTechnology,*
*Hyderabad, India*
*velukantijasnavi21@gmail.com*

*Sapireddy Vaishnavi*
*Neil Gogte Institute ofTechnology,*
*Hyderabad, India*
*svaishnavi1205@gmail.com*

*Gandhe Hasini*
*Neil Gogte Institute ofTechnology,*
*Hyderabad, India*
*hasinigandhe@gmail.com*

***Abstract*--Plant identification plays a crucial role in various fields such as agriculture, herbal medicine, and biodiversity conservation. Traditional AI-based plant identification systems predominantly rely on datasets like Flavia, Folio, and PlantVillage, which feature isolated leaf images captured on uniform backgrounds. These controlled datasets fail to replicate real-world conditions, where images often have diverse backgrounds and feature entire plants. Unlike conventional approaches, our dataset consists of whole plant images with varied backgrounds, such as different soil textures, vehicles, and natural surroundings, making it more representative of real-world scenarios. Our system comprises an intuitive user interface (UI) that enables users to upload images and receive real-time plant classification.**

**Preliminary results demonstrate an accuracy of 90% under these diverse conditions, highlighting the robustness of the model. This research provides an innovative step toward practical and accurate plant identification, paving the way for enhanced applications in agriculture and medicinal research.**

***Keywords—plant identification, Convolutional Neural Networks (CNN), Artificial Intelligence (AI), Computer Vision, Image Classification, Medicinal Plants, Data Augmentation, Real-World Datasets, PyTorch, TensorFlow, Background Variation***

## I. INTRODUCTION

### A. *Objectives of this Research*

The application enables users to upload plant images through an intuitive interface, performs preprocessing to ensure data compatibility with the CNN model, and classifies the plants into predefined categories.

We have two main objectives with this research:

- Develop a model optimized for real-world conditions to overcome limitations of existingmodels which are trained on highly controlled datasets
- To develop a user-friendly UI using front end technologies

## II. LITERATURE SURVEY

Refer TABLE I.

## III. METHODOLOGY

### A. *Dataset Description*

A dataset of plant images taken at a nursery was selected. The dataset consisted of five classes: Aloe Vera, Amla, Brahmi, Neem, and Tulasi, with each class containing 85 images. The original image sizes were 600 × 227, 600 × 283, or 600 × 450.

The dataset was then divided into training and testing sets in an 8:2 ratio.

*Fig. 1 A sample image and its augmented images*

TABLE II. DATA SPLITTING FOR TRAINING AND TESTING

| Set | Data Split (%) | Original Images | Augmented Images | Total Images |
|---|---|---|---|---|
| Train | 80 | 340 | 680 | 1020 |
| Test | 20 | 85 | 170 | 255 |
| Total | 100 | 425 | 850 | 1275 |

**Table II** shows the 80:20 split of 1,275 images into trainingand testingsets, including both original and augmented data.

### B. *Image Pre-Processing*

1) The images were resized to a standard size of 64×64[1] using OpenCV.
2) The pixel values of the images were normalized to range between 0 and 1.

### C. *Data Augmentation*

To increase the dataset size, data augmentation was performed. The dataset was expanded threefold, from 425 images to 1,275 images. Each image was rotated 90 degrees in a clockwise direction and flipped along the vertical axis. The final dataset consisted of 1,020 training images and 255 testing images.

### D. *Model Architecture*

1) *ANN Architecture*
   a) *First Dense Layer:* Accepts RGB images of size 64x64 with 3 channels (color channels: Red, Green, Blue) after flattening. Number of neurons is 768. Activation function (ReLU / LeakyReLU / Tanh) is applied for non-linearity.
   b) *Second Dense Layer:* Accepts an input array of size 768. The number of neurons is 96.Activation function (ReLU / LeakyReLU / Tanh) is applied for non-linearity.

TABLE I: LITERATURE SURVEY WITH EXISTING WORKS.

| Authors | Dataset | Model | Methodology | Summary |
|---|---|---|---|---|
| **Bisen, D.[1],2021** | Dataset of plant images | CNN (Convolutional, Pooling, Fully Connected, RELU, Dropout) | Data augmentation to increase size of data set. | Deep CNN-based plant species recognition through leaf features. |
| **R. Azadnia et al [2], 2022** | Dataset of 750 expanded to 13,500 images using data augmentation | CNN with GAP and Dense layers | A more generic dataset is used to train the model | CNN-based medicinal plant identification using GAP for classification.<br><br>Global Average Pooling replaces fully connected layers to reduce overfitting. |
| **Wagle et al [3], 2021** | PlantVillage (9 species), Flavia (32 species) | CNN(Convolutional Layers, Pooling layers, fully connected layers, activation function and output layer) | Use dataset of plants instead of leaves. | Proposes compact CNN models (N1, N2, N3) for plant leaf classification<br><br>It aims to create a computationally compact model. |
| **B. R. Pushpa and N. S. Rani [4], 2023** | Includes 40 Ayurvedic medicinal plant species. | A lightweight CNN optimized for accurate plant classification. | Improve robustness and image variations. | Ayur-PlantNet identifies 40 Ayurvedic medicinal plant species using leaf images.<br><br>It provides efficient plant identification while preserving traditional knowledge. |

**Table I** presents a literature survey of existing works on plant identification, summarizing datasets, models, and key methodologies used by various researchers in the field.

c) *Third Dense Layer:* Accepts an input array of size 96. The number of neurons is 96.

d) *Loss Function:* Utilizes the Softmax with Cross-Entropy loss function to compute the error

e) *Optimizer:* We use vanilla gradient descent or mini-batch gradient descent.

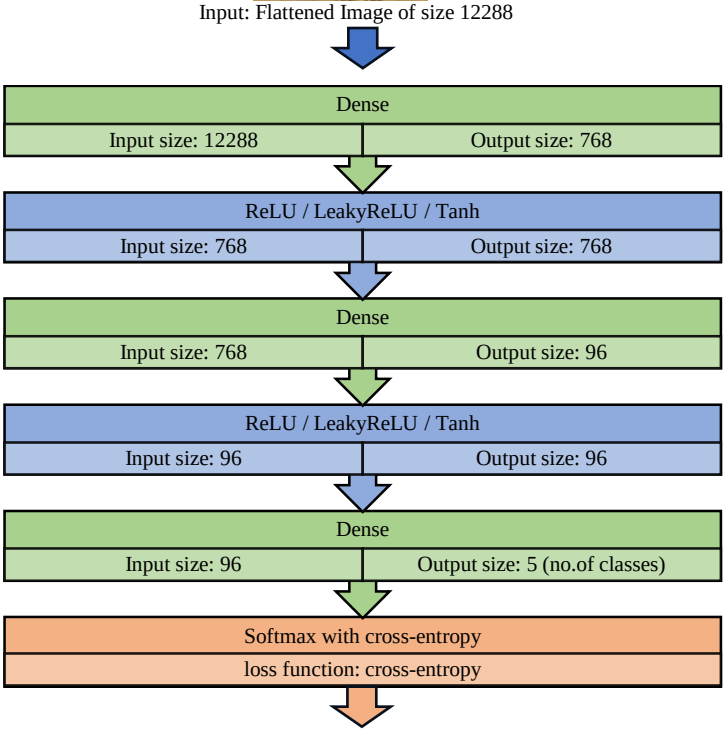



*Fig. 2 ANN Architecture*

1) *CNN Architecture*

a) *First Convolutional Block:*

- *Convolution Layer:* Uses 32 filters of size 3x3, stride of 1, and padding of 1 to preserve spatial dimensions, producing an output of size 64x64x32.
- *Activation Function (ReLU / LeakyReLU / Tanh):*Introduces non-linearity to the network.
- *MaxPooling Layer:* Reduces the spatial dimensions to 32x32x32 with a pooling size of 2x2 and stride of 2.

b) *Second Convolutional Block:*

- *Convolution Layer:* Applies 64 filters of size 3x3, stride of 1, and padding of 1, producing an output of size 32x32x64.
- *Activation Function (ReLU / LeakyReLU / Tanh):*Introduces non-linearity.
- *MaxPooling Layer:* Reduces the spatial dimensions to 16x16x64 using a pooling size of 2x2 and stride of 2.

c) *Flatten Layer:* Converts the 3D tensor of size 16x16x64 into a 1D vector of size 16384.

d) *Fully Connected Layers:*

- *Dense Layer (FC1):* Contains 64 nodes and takes the flattened vector as input. ReLU is applied for non-linearity.
- *Dense Layer (Output):* Contains 5 nodes corresponding to 5 classes (medicinal plant categories). This layer outputs class probabilities.

e) *Loss Function:* Utilizes the Softmax with Cross-Entropy loss function to compute the error

f) *Optimizer:* We use vanilla gradient descent or mini-batch gradient descent. For our final model we used Adam optimizer.

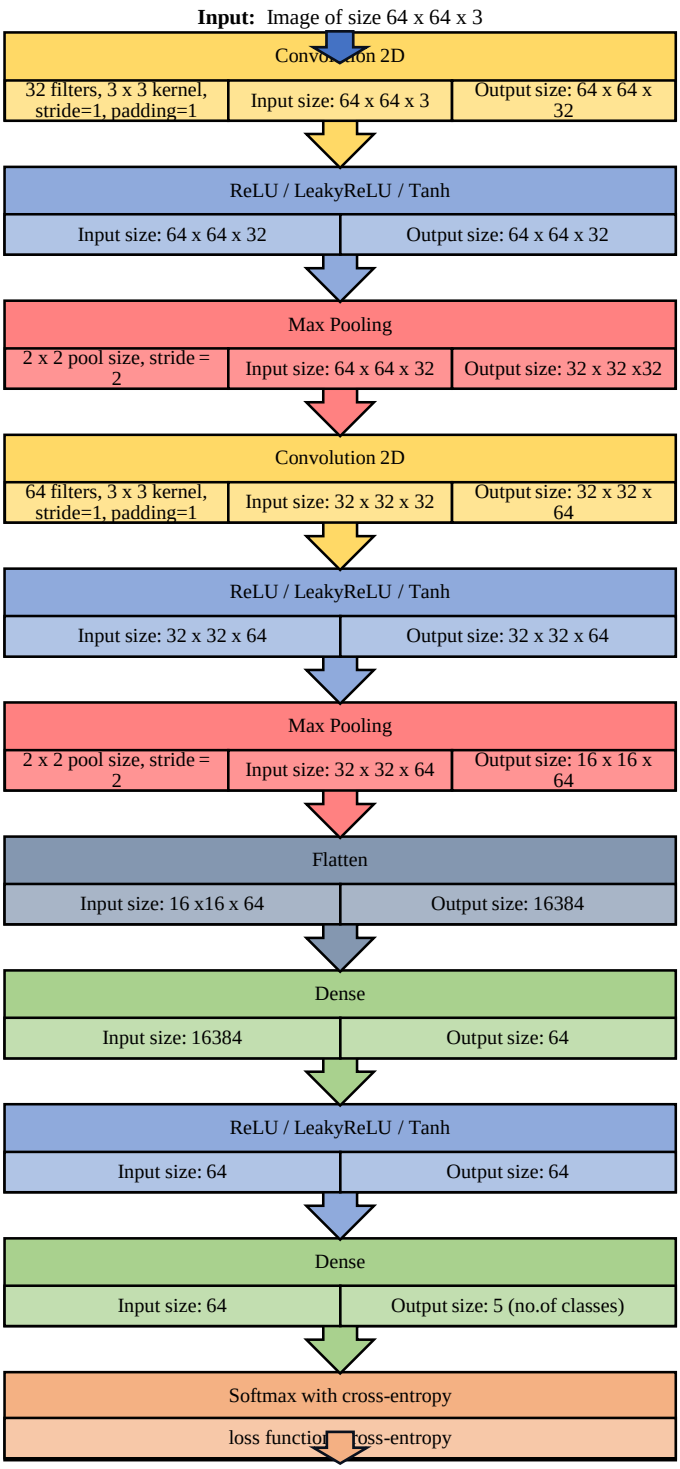



*Fig. 3 CNN Architecture*

### A. *Types of Models*

#### 1) *Scratch Model*

- Building a CNN without using deep learning frameworks like TensorFlow or PyTorch.
- Requires implementing all components manually, such as convolution operations, activation functions, backpropagation, and gradient descent using NumPy
- Provides full control over computations, making it ideal for learning how CNNs work internally but is computationally expensive

#### 2) *TensorFlow Model*

- TensorFlow is a Google-backed deep learning framework designed for scalability and deployment.
- Uses Keras API for building CNNs with simple, high-level functions like Conv2D, MaxPooling2D, and Dense.
- Best for quick prototyping, large-scale training, and deployment.

#### 3) *PyTorch Model*

- PyTorch is a Facebook-backed deep learning framework that offers dynamic computation graphs, making debugging easier.
- Uses an object-oriented approach, defining CNNs as classes that inherit from torch.nn.Module.
- Provides flexibility for research and experimentation with lower-level control over computations than TensorFlow.

TABLE III. COMPARISON OF NUMPY vs TENSORFLOW vs PYTORCH

| **Feature** | **NumPy Model** | **TensorFlow** | **PyTorch** |
|---|---|---|---|
| **Ease of Use** | Hard | Easy | Moderate |
| **Flexibility** | Full control | Moderate | High |
| **Performance** | Slow (CPU-based) | Fast | Fast |
| **Debugging** | Hard | Moderate | Easy |
| **Best For** | Learning CNN internals | Quick prototyping & deployment | Research & experimentation |

**Table III**compares NumPy, TensorFlow, and PyTorch across key features

For each of these models, we chose to compare them on the following parameters:

- Optimizer Comparison
  - Optimizer: Vanilla, Epochs: 15, Learning Rate: 0.005, Activation: LeakyReLU
  - Optimizer: Mini batch, Epochs: 15, Learning Rate: 0.005, Activation: LeakyReLU
- Activation Function Comparison
  - Optimizer: Mini batch, Epochs: 15, Learning Rate: 0.005, Activation: LeakyReLU
  - Optimizer: Mini batch, Epochs: 15, Learning Rate: 0.005, Activation: ReLU
  - Optimizer: Mini batch, Epochs: 15, Learning Rate: 0.005, Activation: Tanh
- Epochs Comparison
  - Optimizer: Mini batch, Epochs: 10, Learning Rate:0.005, Activation: LeakyReLU
  - Optimizer: Mini batch, Epochs: 15, Learning Rate: 0.005, Activation: LeakyReLU
- Learning Rate Comparison
  - Optimizer: Mini batch, Epochs: 15, Learning Rate: 0.005, Activation: LeakyReLU
  - Optimizer: Mini batch, Epochs: 15, Learning Rate: 0.01, Activation: LeakyReLU

*B. Technical Stack*

*1) Front-End*

*a) HTML*: Structures the web-based user interface for the application.

*b) JavaScript:* Adds interactivity for image uploads and dynamic displays of classification results.

*2) Back-End*

*a) Python*: Implements the CNN model and handles backend processing.

*b) Python Libraries*:

- *NumPy:* For numerical computations in model development.
- *OpenCV:* For preprocessing and handling plant images.
- *PyTorch:* For building, training, and deploying the CNN model.
- *Pillow (PIL)*: For basic image manipulation.
- *Matplotlib:* For visualizing model training metrics.

*c) Pickle:* For saving and loading serialized objects like the trained CNN model.

*3) Linking*

*a) Flask* bridges the frontend and backend by providing APIs for image upload and classification results.

*C. User Interface*

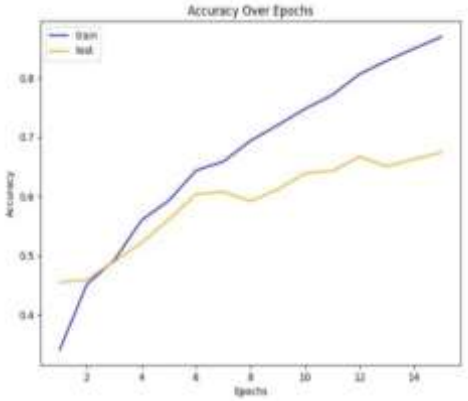
A user-friendly interface was designed and integrated with the pre-trained model. The interface allows end-users to upload an image from their local system, which undergoes pre-processing to resize it to the required dimensions of 64 × 64 pixels and normalize the pixel values. The pre-processed image is then passed to the pre-trained model for inference, and the predicted class label is obtained. Finally, the interface displays both the uploaded image and the corresponding predicted class to the user.


Fig. 4 User Interface(UI)

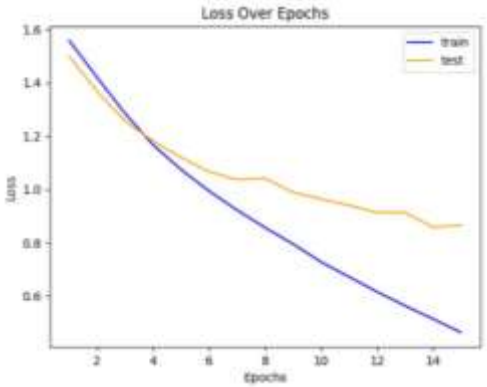
## IV. RESULTS AND DISCUSSIONS

For our comparison of NumPy, TensorFlow and PyTorch models we have described the observed accuracies in TABLEIV

ANN PyTorch:



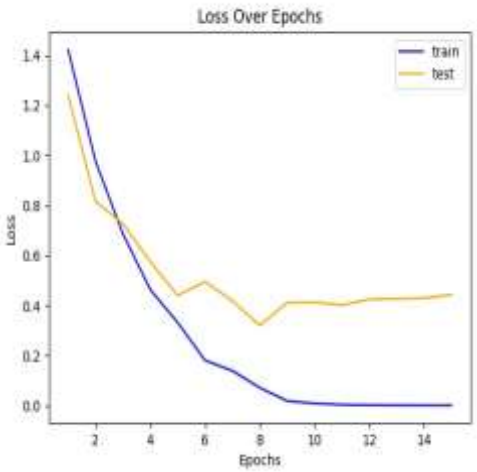

(a)



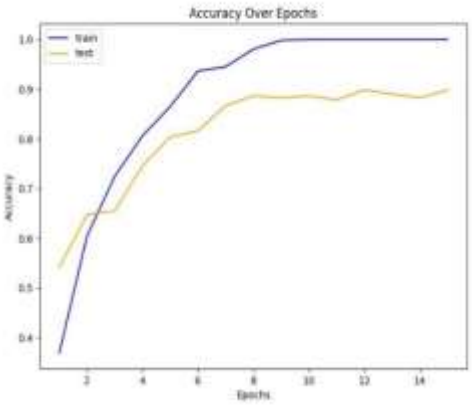

(b)

*Fig.5 Mini-Batch Gradient Descent with LeakyReLU – Epochs 15 – LR 0.01 (a) Accuracy performance (b) Loss performance*

CNN PyTorch:




(a)




(b)

*Fig. 6 Mini-Batch Gradient Descent with LeakyReLU – Epochs 15 – LR 0.01 (a) Accuracy performance (b) Loss performance*

From Table IV, it is clear that a CNN PyTorch modelhas the best performance for the architecture being used. Therefore, for our final model we chose a CNN PyTorch model with Adam optimizer.

The model was trained with a learning rate of 0.001 for 10 epochs. The activation function used is LeakyReLU. Analysis of the confusion matrices indicates some misclassification patterns, specifically between the classes Tulasi and AloeVera with 6 AloeVera images identified as Tulasi and 8 Tulasi images identified as AloeVera. The accuracy of our model is 89.4%.

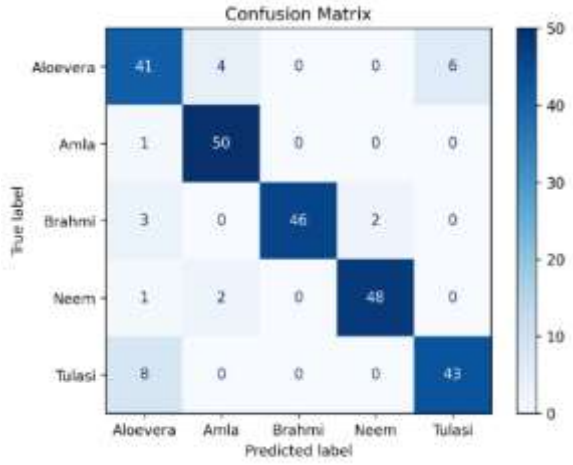
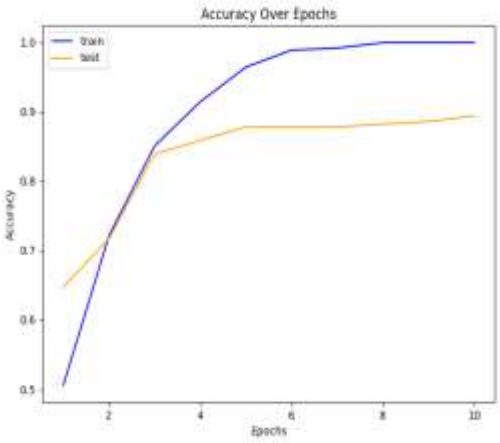
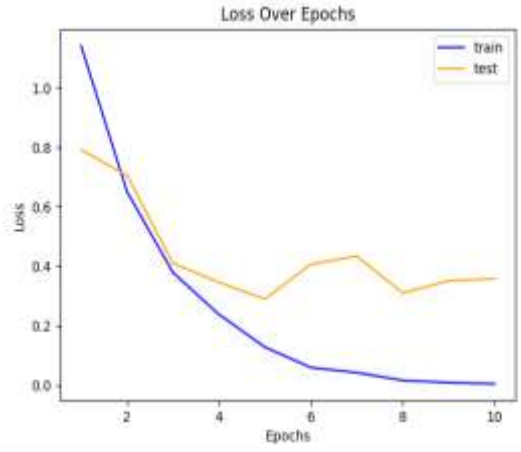



*Fig. 7 Confusion matrix for 10 epochs*




(a)




(b)

*Fig. 8Accuracy and loss performance in the training and testing (a) Accuracy performance (b) Loss performance*

We also tested the final model with one extra convolution layer and more filters. The accuracy with an extra convolutional layer was only 80% and the accuracy with 64 filters in first and second convolutional layer was only 86.27%.

TABLE IV. TABLE TO COMPARE ACC

| **Model Specification** | **ANN NumPy** | | **CNN NumPy** | | **ANN Tenso** |
|---|---|---|---|---|---|
| | *Train* | *Test* | *Train* | *Test* | *Train* |
| **Vanilla Gradient Descent with LeakyReLU – Epochs 15 – LR 0.005** | 29.21 | 25.49 | 20.0 | 20.0 | 36.3 |
| **Mini-Batch Gradient Descent with LeakyReLU – Epochs 15 – LR 0.005** | 56.58 | 47.95 | 38.75 | 22.72 | 34.4 |
| **Mini-Batch Gradient Descent with LeakyReLU – Epochs 10 – LR 0.005** | 53.29 | 56.59 | 39.14 | 21.21 | 28.5 |
| **Mini-Batch Gradient Descent with LeakyReLU – Epochs 15 – LR 0.01** | 56.68 | 44.39 | - | - | 29.1 |
| **Mini-Batch Gradient Descent with ReLU – Epochs 15 – LR 0.005** | 57.75 | 48.03 | 20.44 | 20.45 | 37.5 |
| **Mini-Batch Gradient Descent with Tanh – Epochs 15 – LR 0.005** | 57.65 | 44.77 | 29.06 | 25.75 | 36.8 |

**Table IV** summarizes the performance of various ANN and CNN models. The C accuracy of 89.8%

Our model still has some limitations:

- The accuracy is only 89.4% and can still be improved
- All the images of our data set are taken from a nursery, focusing exclusively on young plants, which limits the model's ability to identify older plants or those in varied growth stages
- Out data set only has 1275 images, with only 5 classes, to compare our model with other similar models, we need a larger data set

TABLE V.PERFORMANCE METRICS

| Plant Species | Accuracy | Precision | Sensitivity (Recall) | Specificity | AUC | F1-Score |
|---|---|---|---|---|---|---|
| AloeVera | 0.909 | 0.76 | 0.80 | 0.93 | 0.87 | 0.78 |
| Amla | 0.972 | 0.89 | 0.98 | 0.97 | 0.97 | 0.93 |
| Brahmi | 0.980 | 1.00 | 0.90 | 1.00 | 0.95 | 0.95 |
| Neem | 0.980 | 0.96 | 0.94 | 0.99 | 0.96 | 0.95 |
| Tulasi | 0.945 | 0.88 | 0.84 | 0.97 | 0.90 | 0.86 |
| **Average per class** | 0.957 | 0.89 | 0.89 | 0.97 | 0.93 | 0.89 |

**Table V** shows strong model performance, with highest accuracy (98%) for Brahmi and Neem, and an overall average accuracy of 95.7% and F1-score of 89%.

## V. CONCLUSION

Our research takes a unique approach to plant identification by addressing real-world challenges often overlooked in existing AI models. Most plant identification systems rely on datasets such as Flavia, Folio, and PlantVillage, which feature a wide variety of leaf images captured on uniform backgrounds (typically white) with isolated leaves. These controlled conditions fail to reflect the complexities of real-world scenarios.

In the paper "Efficient and Automated Herbs Classification Approach Based on Shape and Texture Features using Deep Learning"[5] by Amgad Muneer and colleagues, the authors analyzed the impact of background variation on accuracy. They reported a model accuracy of 93% for white backgrounds, which dropped to 91.75% for orange backgrounds and significantly declined to 59% for floral backgrounds. This demonstrates the challenges posed by varied backgrounds in practical applications.

Our research addresses this issue by utilizing images of entire plants captured in a nursery environment. These images include diverse backgrounds such as different soil textures, vehicles, and other objects, making the dataset more representative of real-world conditions. Additionally, we havedeveloped a user-friendly interface that allows users to upload an image and receive the predicted plant class.

The accuracy of our model 90% which less as compared to other similar models but the response time of the model is much faster than in other similar models. This is due to the very simple architecture of our model. It is also much smaller in size compared other models.

We can expand this research by adding features like:

- Increase number of classes and dataset size
- Integrate live upload of images from phone
- Integration with geographic information system

## REFERENCES


1. D. Bisen, “Deep convolutional neural network based plant species recognition through features of leaf,” *Multimedia Tools and Applications*, vol. 80, pp. 31677–31690, October 2020.
2. R. Azadnia, M. M. Al-Amidi, H. Mohammadi, M. A. Cifci, and E. Cavallo, “An AI-based approach for medicinal plant identification using deep CNN based on global average pooling,” *Agronomy*, vol. 12, no. 11, article 2723, November 2022.
3. S. A. Wagle, R. Harikrishnan, S. H. M. Ali, and M. Faseehuddin, “Classification of plant leaves using new compact convolutional neural network models,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 5, pp. 123–130, 2021.
4. B. R. Pushpa and N. S. Rani, “Ayur-PlantNet: an unbiased light weightdeep convolutional neural network for Indian Ayurvedic plant speciesclassification,” J. Appl. Res. Med. Aromat. Plants, vol. 34, 2023.
5. A. Muneer and S. Mohamed, “Efficient and automated herbs classification approach based on shape and texture features using deep learning,” *IEEE Access*, vol. 9, pp. 123456–123465, 2021.
6. T. S. Priyadarshini, M. A. Hameed and S. A. Qadeer, "NDeep Learning Heart Stroke Prediction Model Integration of MMAM with NB(MMAM-NB) and DT(MMAM-DT)," *2023 IEEE 5th International Conference on Cybernetics, Cognition and Machine Learning Applications (ICCCMLA)*, Hamburg, Germany, 2023, pp. 373-380, doi: 10.1109/ICCCMLA58983.2023.10346833.
7. H. Puri, J. Chaudhary, K. R. Raghavendra, R. Mantri and K. Bingi, "Prediction of Heart Stroke Using Support Vector Machine Algorithm," *2021 8th International Conference on Smart Computing and Communications (ICSCC)*, Kochi, Kerala, India, 2021, pp. 21–26.
8. N. Sharma, M. K. Mishra, J. S. Chadha and P. Lalwani, "Heart Stroke Risk Analysis: A Deep Learning Approach," *2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT)*, Bhopal, India, 2021, pp. 543-598, doi: 10.1109/CSNT51715.2021.9509665.
9. D. H. Wolpert, “Stacked generalization,” *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.
10. B. T. Mashi, M. Hamada, J. J. Tanimu, P. Robert and T. J. Samson, "An Ensemble Approach for Stroke Prediction," *2024 IEEE 17th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, Kuala Lumpur, Malaysia, 2024, pp. 381-388, doi: 10.1109/MCSoC64144.2024.00069.
11. G. S. Sailasya , G. L. A. Kumari, "Analyzing the Performance of Stroke Prediction Using ML Classification Algorithms," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 6, 2021.
12. M. Wang, X. Yao and Y. Chen, "An Imbalanced-Data Processing Algorithm for the Prediction of Heart Attack in Stroke Patients," *IEEE Access*, vol. 9, pp. 25394–25404, 2021, doi: 10.1109/ACCESS.2021.3057693.
13. N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.
14. F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
15. G. Biau, "Analysis of a Random Forests Model," *Journal of Machine Learning Research*, vol. 13, pp. 1063–1095, 2012.
16. K. Pearson, "LIII. On Lines and Planes of Closest Fit to Systems of Points in Space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.
17. L. van der Maaten and G. Hinton, "Visualizing Data Using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 11, pp. 2579–2605, 2008.
18. Scikit-learn Developers, “Ensemble methods,” *Scikit-learn: Machine Learning in Python*, [Online]. Available: https://scikit-learn.org/stable/modules/ensemble.html.
19. P. Kunwar and P. Choudhary, "A Stacked Ensemble Model for Automatic Stroke Prediction Using Only Raw Electrocardiogram," *Intelligent Systems with Applications*, vol. 17, p. 200165, 2023.
20. P. Chakraborty et al., "Predicting Stroke Occurrences: A Stacked Machine Learning Approach with Feature Selection and Data Preprocessing", *BMC Bioinformatics*, vol. 25, no. 1, p. 329, 2024.