A Web based Deep Learning for Medicinal Plant Identification

By

No.	Names	Index
1	**Kon James Ayuen**	**23 - ML -005**
2	**Malith Dut Malual Wol**	**23 - CDA -022**
3	**Akot Deng Akot**	**23 -ML -002**
4	**Biar Chagai Atem**	**23 - CDA -010**
5	**Deng Peter Nyuon**	**23 - AI -009**

University of Juba

Department of Computer Science & Information Technology

Lecturer's Name: **Ustaz. Thon Malek Garang Ok**

November, 2025

Table of Contents

Abstract

Traditional medicinal systems, on which up to 80% of the population in developing countries relies, face a critical challenge in accurate plant identification. Reliance on specialist knowledge is time-consuming and prone to human error, especially when plant species exhibit visual similarities. This project addresses this need by designing, developing, and deploying an end-to-end deep-learning-powered system for medicinal plant identification.

The methodology involved training and evaluating multiple Convolutional Neural Network (CNN) architectures—including MobileNetV2, EfficientNetB7, and ResNet50—on the "Indian Medicinal Leaves" dataset, utilizing transfer learning. The ResNet50 model emerged as the top performer, achieving a Test Accuracy of 94.5% and a Macro Average F1-Score of 94.5%. Detailed analysis confirmed excellent performance for Lemon and Neem classes (F1-Scores of 0.97 and 0.98, respectively), while identifying the Mango class (F1-Score 0.87) as the primary focus for future refinement.

The system is deployed via a lightweight Flask web application, structured as a three-tier architecture, coordinating image input, model inference, and data retrieval. A core feature is the integration of Grad-CAM visualization, which generates heatmaps to explain model predictions by highlighting biologically relevant leaf features (venation and texture), thereby enhancing transparency and user trust. The system's use of lightweight models ensures acceptable inference latency (20–50 ms with GPU acceleration) suitable for real-time web and potential edge deployment. This work successfully provides a scalable, accurate, and explainable digital tool to support herbalists, researchers, and community members in the reliable classification of medicinal leaves.

1. Introduction

1.1. Background on medicinal plants and motivation for automated identification.

Medicinal plants have been at the heart of traditional healing systems for thousands of years. Across Africa, Asia, and many parts of the global South, communities continue to rely heavily on herbal remedies for treating common illnesses, promoting wellness, and preserving cultural knowledge. The World Health Organization (WHO) estimates that up to 80% of the population in developing countries depend on plant-based medicine as their primary source of healthcare (WHO, 2023). These plants are not just biological resources—they represent local identity, history, and collective knowledge passed down through generations.

Despite their importance, correctly identifying medicinal plants remains a challenge. Traditional identification relies on experienced botanists, herbalists, or field experts who examine physical characteristics such as leaf texture, vein patterns, color variations, and plant morphology. While this method is effective, it is time-consuming, heavily dependent on expert availability, and prone to human error, especially when species share similar visual traits (Mayfield et al., 2021). Misidentification can lead to ineffective treatment or even harmful health outcomes.

As technology rapidly advances, there is a growing need to preserve indigenous knowledge while providing modern tools that make plant identification more accessible. Deep learning—particularly convolutional neural networks (CNNs)—has emerged as a promising solution. CNNs can learn subtle visual patterns in images, enabling them to classify plants with a level of precision that rivals human experts (Szegedy et al., 2016). When such a model is integrated into a user-friendly web application, community members, students, researchers, agriculture officers, and herbal practitioners can quickly identify medicinal plants using a simple leaf image captured with a smartphone.

Automating medicinal plant identification is therefore not just a technical exercise—it is a contribution toward bridging the gap between traditional knowledge and modern digital tools. By harnessing the capabilities of deep learning and pre-trained models, we aim to overcome the limitations of traditional plant identification methods and provide a scalable, efficient, and accurate solution for identifying medicinal plants. It empowers communities, supports conservation efforts, enhances agricultural decision-making, and provides a foundation for safer and more accessible use of plant-based medicine. This project responds to these needs by

developing a deep-learning-powered medicinal plant classification system and deploying it through a lightweight Flask-based web application.

The system brings together artificial intelligence, mobile accessibility, and botanical knowledge to support real-world use in both rural and academic contexts.

1.2. Problem statement and specific project objectives.

1.2.1 Problem Statement

Although medicinal plants play an essential role in healthcare and local culture, many people lack access to expert knowledge needed to correctly identify them (Mayfield et al., 2021). Leaf-based identification depends heavily on specialists, and mistakes can occur when species share similar appearance or when images are captured under poor lighting or at unusual angles (Jones et al., 2020). Current digital resources are either too technical or not tailored to local medicinal species (Wäldchen & Mäder, 2018). There is a clear need for an automated, accurate, and user-friendly system that can assist both experts and everyday users in identifying medicinal plants from simple leaf images.

1.2.2 Main Objective

To design, develop, and deploy an end-to-end deep learning–powered medicinal plant identification system that accurately classifies plant species from leaf images and delivers real-time, explainable predictions through a Flask-based web application

1.2.3 Specific Objectives

1. Develop a complete deep learning pipeline for classifying medicinal plant species from leaf photographs.
2. Collect, clean, and preprocess a dataset of medicinal plant images sourced from Kaggle and local field collections.
3. Design and evaluate multiple CNN and transfer learning models—including MobileNetV2, ResNet50, and EfficientNet—to identify the best-performing architecture.
4. Incorporate explainability tools such as Grad-CAM to visualize which leaf regions influenced each model prediction, ensuring transparency and trustworthiness.
5. Build and deploy a Flask web application capable of real-time prediction, confidence scoring, and retrieval of relevant medicinal information for each species.

6. Document the entire workflow, including methods, experiments, evaluation results, system design, and deployment, in line with academic standards.

2. Literature Review

2.1. Overview of prior work on plant/leaf classification, transfer learning in plant ID, and explainability for botanical models.

2.1.1 Leafsnap: early mobile leaf recognition

(Kumar et al., 2012) introduced Leafsnap, one of the first practical attempts to bring automated leaf identification to mobile users. The system combined careful image segmentation, contour and shape descriptors, and classical classification methods to recognize tree species from single-leaf images. Leafsnap demonstrated that mobile-based plant ID was feasible, but also exposed key limitations: the approach required fairly clean, well-centered leaves and struggled with complex backgrounds and in-field variability. For our medicinal-plant project, Leafsnap is important historically — it shows the value of segmentation and domain-aware preprocessing, and it motivates the move to deep learning which better handles real-world noise.

2.1.2 PlantCLEF / ImageCLEF benchmarks: standardized challenges

The ImageCLEF/PlantCLEF initiatives created large, annotated datasets and shared tasks for plant identification, catalyzing research by offering common benchmarks and emphasizing "in-the-wild" images (different viewpoints, backgrounds, and organs). Results from these challenges highlighted that good performance on curated datasets does not always transfer to field conditions, and that metadata (location, date) can materially help disambiguate visually similar species. For our work, PlantCLEF findings argue for combining image features with simple metadata where available, and for evaluating models on diverse, realistic samples. (Goëau et al., 2013)

2.1.3 Deep residual learning (ResNet) and its impact

ResNet architecture tackled the degradation problem in deep networks via residual connections, enabling much deeper, more accurate models. In plant classification tasks, ResNet variants became a go-to backbone because they balance ease of fine-tuning with strong feature extraction. For medicinal-plant classification, ResNet-based transfer learning often yields excellent accuracy when fine-tuned on domain images—especially for datasets with moderately large sample sizes per class. However, their size and compute demand must be weighed against deployment constraints. (He et al., 2016)

2.1.4 EfficientNet: scaling networks efficiently

EfficientNet proposed a principled way to scale network width, depth, and resolution together, achieving state-of-the-art accuracy with fewer parameters. In botanical image tasks, EfficientNet variants (B0–B7 etc.) often offer an excellent trade-off between accuracy and inference cost, making them attractive for prototyping and deployment. Our experiments (and literature) show EfficientNet is particularly effective when computational resources are limited but high accuracy is still required. (Tan & Le, 2019)

2.1.5 MobileNetV2: lightweight models for edge/mobile

MobileNetV2 introduced inverted residuals and linear bottlenecks, enabling compact, fast models ideal for mobile or browser-based inference. Several plant-ID studies show that MobileNetV2 is competitive when latency and memory are important (e.g., mobile field apps). For this medicinal-plant system, MobileNetV2 is a prime candidate for on-device or low-cost cloud deployments where responsiveness matters. The trade-off is modest accuracy loss versus larger architectures. (Sandler et al., 2018)

2.1.6 Explainability with Grad-CAM

Grad-CAM (Gradient-weighted Class Activation Mapping) produces intuitive heatmaps that show which image regions most influenced a CNN's decision. In plant science, Grad-CAM is widely used to verify that models attend to biologically meaningful features (vein patterns, margins) rather than background artifacts. Integrating Grad-CAM into a user-facing app helps build trust with herbalists and end-users—letting them see *why* a prediction was made. For this project, Grad-CAM is a core explainability tool that supports both model debugging and community acceptance. (Selvaraju et al., 2017)

2.1.7 Surveys on machine learning for species ID

Wäldchen and Mäder provided a thoughtful survey of machine-learning approaches for species identification, comparing classical features to modern deep-learning methods and discussing typical failure modes (class imbalance, domain shift). They stress evaluation on realistic datasets and the importance of explainability and human expert involvement. This survey helps frame our methodological choices: prefer transfer learning for small datasets, use augmentation and class-weighting for imbalance, and rely on explainability tools to validate model focus. (Wäldchen & Mäder, 2018)

2.1.8 Transfer learning analyses specific to plant datasets

Kaya and Uğur empirically analyzed transfer learning for plant recognition, comparing multiple pre-trained backbones and fine-tuning strategies. Their findings reinforce that pre-trained models adapted with moderate fine-tuning outperform small models trained from scratch—particularly when per-class image counts are limited. They also note that architecture choice depends on deployment goals: smaller models for mobile use, larger ones for accuracy. Their work directly informs our plan to compare MobileNetV2, ResNet50, and EfficientNet variants. ((Kaya & Uğur, 2019))

2.1.9 Going deeper with herbarium specimens

Carranza-Rojas and colleagues examined identification of herbarium specimens using deep architectures and large, digitized collections. Their results show deep models can excel when trained on high-quality, labeled specimen images and that model performance depends on curation and label consistency. While herbarium images differ from field leaf photos, their work emphasizes the value of large curated datasets and careful metadata—lessons that apply to building a robust medicinal-plant dataset combining field photos and public repositories. (Carranza-Rojas et al., 2017)

2.1.10 Practical evaluation of plant-ID apps and real-world limits

Jones et al. performed a pragmatic evaluation of popular plant identification apps using realistic images. They found that while apps are useful for common species and clear photographs, accuracy drops for less common plants, damaged specimens, or pictures taken under bad conditions. The study underscores the importance of domain adaptation, inclusion of local species, and communicating uncertainty to users. For our medicinal-plant app, this work motivates (a) providing top-k suggestions rather than a single label, (b) showing confidence scores and Grad-CAM maps, and (c) allowing users to submit corrections to improve the model over time. (Jones et al., 2020)

3. Methodology

3.1. Dataset description and collection methodology.

The dataset used for this project is named "Indian Medicinal Leaves", sourced from Kaggle, a widely recognized platform for datasets containing high-resolution images of various medicinal plant leaves native to India. This dataset comprises images of medicinal plants, specifically focusing on their leaves. The dataset includes 1,500+ images across 30+ plant species, each labeled with its botanical name, each represented by a separate subfolder within the dataset directory. These plants are known for their therapeutic properties in traditional Indian medicine systems like Ayurveda and Siddha. Leaves were collected from botanical gardens, herbal farms, and forest regions across India, with a focus on plants used in Ayurveda and traditional healing.

Leaves were placed under natural lighting conditions with varying backgrounds on neutral backgrounds (white or black) to reduce noise, simulating real-world scenarios. Images were taken using mobile cameras or DSLR setups under natural or controlled lighting.

This variability enhances the robustness of the dataset, allowing models trained on it to generalize better to real-world scenarios. The dataset was split into 70% training, 15% validation, and 15% testing, ensuring class balance across splits and randomness.

3.2. Preprocessing and augmentation techniques.

To prepare the dataset for training, the following preprocessing steps were applied:

- Resizing: All images were resized to 224×224 pixels to standardize input dimensions for CNNs.
- Normalization: Pixel values were scaled to the [0, 1] range.
- Label Encoding: Class labels were converted to one-hot vectors for multi-class classification.

To improve model robustness and reduce overfitting, data augmentation was applied using: Data augmentation was implemented to improve model generalization, incorporating random horizontal and vertical flips and random rotations.

- Random rotations (±20°)
- Horizontal and vertical flips

- Zooming (up to 20%)
- Brightness and contrast adjustments

These augmentations simulate real-world variability in leaf orientation, lighting, and scale.

3.3. Model architectures explored and rationale for choices.

Several deep learning architectures were explored:

- Baseline CNN: The network contains **3 Convolutional (Conv2D) layers**
- Transfer Learning Models:
    - MobileNetV2: Lightweight and efficient, ideal for deployment on edge devices.
    - ResNet50: Deep residual learning to handle complex leaf textures and vein patterns.
    - EfficientNetB7: Balances accuracy and computational efficiency.

Transfer learning was chosen due to the limited dataset size and the models' ability to leverage pre-trained features from ImageNet, which accelerates convergence and improves accuracy.

3.4. Training procedure and hyperparameter selection.

Models were trained using TensorFlow/Keras with the following configuration:

- Loss Function: Categorical Cross-Entropy
- Optimizer: Adam with an initial learning rate of 0.0001
- Batch Size: 16
- Epochs: 15, with early stopping based on validation loss
- Dropout: 0.5 in fully connected layers to prevent overfitting
- Learning Rate Scheduler: ReduceLROnPlateau to adapt learning rate dynamically

Hyperparameters were tuned using grid search over learning rates, dropout rates, and batch sizes.

3.5. Evaluation metrics and experimental setup.

Model performance was evaluated using:

- Accuracy: Overall classification correctness

- Precision, Recall, F1-score: To assess per-class performance, especially for underrepresented species
- Confusion Matrix: To visualize misclassifications and identify confounding classes
- Top-3 Accuracy: Useful in real-world applications where multiple suggestions are acceptable

Experiment was conducted on Kaggle with a Tesla T4 GPU, ensuring reproducibility via fixed random seeds and consistent data splits. Each model was trained and evaluated three times, and average metrics were reported.

4. System Design

The Medicinal Plant Classification system is designed as a clean and well-structured three-tier architecture. Each layer plays a unique role: the Presentation Layer (HTML/CSS) handles user interaction, the Application Logic Layer (Flask) manages communication and decision-making, and the Data/Modeling Layer (TensorFlow/Keras) performs the intelligence behind the scenes. Together, they form a smooth pipeline that guides a simple leaf image all the way to a meaningful medicinal plant diagnosis.

4.1 High-Level Architecture and Data Flow

At its core, the system functions like a conversation between the user and the machine. A user submits a photo, and the system—patiently and systematically—transforms that raw image into a confident classification accompanied by scientific insight. The architecture follows a synchronous request–response style, meaning every user action triggers immediate processing on the server before a result is returned.

Data Flow Sequence

1. Input (Client – User Interaction)
   Everything begins with a user opening the web app. Whether they upload an image from their computer or capture one directly from their camera, the user's action sends the system into motion. This image becomes the raw input from which the model will try to understand the plant's identity.

2. Inference Trigger (Flask – app.py)
   Once the user submits the image, the Flask backend springs into action. The file is received through a POST request, stored safely under static/uploads/, and its file path

is passed along to the model pipeline. This ensures the image is both preserved and accessible for further processing.

3. Preprocessing (Model Utilities)
   Before the model can make sense of the image, it needs to be "prepared." The predict_species() function handles this. Using OpenCV, the image is read, converted from BGR to RGB, and resized neatly to 224 × 224 pixels, exactly what the ResNet50 architecture expects. Finally, the image is normalized using preprocess_input() so that the model interprets it correctly.

4. Prediction (Deep Learning Model)
   Now the processed image is ready for intelligence. It is passed through the preloaded ResNet50-based model. The model outputs probabilities for each plant class. The system selects the top prediction, sorts it by confidence, and maps it back to human-readable labels using the saved class_indices.pkl file.

5. Explainability (Grad-CAM Visualization)
   To give users more than just a prediction, the system adds a layer of transparency. The generate_gradcam() function is called to produce a Grad-CAM heatmap—essentially a visualization that highlights which part of the leaf the model paid attention to. This heatmap is saved under static/explanations/ and later shown to the user.

6. Data Retrieval (Medicinal Information)
   Once the top species label is identified, the system retrieves rich descriptive information about the plant through medicinal_data.py, which stores names, scientific classifications, and medicinal uses. This transforms a simple prediction into meaningful knowledge.

7. Output (Client – User View)
   Finally, all these components come together. Flask renders result.html, presenting the prediction, confidence scores, plant information, and both the original and Grad-CAM images. The user sees not only *what* plant it is, but also *why* the model thinks so and *how* the plant is used medicinally.

4.2 Flask Application Components and API Endpoints

The Flask application acts like the system's "traffic controller," coordinating every stage—from receiving the image to returning an analysed result. It ensures that each layer of the architecture communicates seamlessly and that errors are handled gracefully.

a. Flask App Components

Component	File	Description
Main Application	app.py	The heart of the system. It initializes Flask, sets up file paths, and defines the behaviour of every route. Every request passes through here.
Model Backend	utils/model_utils.py	This is the machine's "thinking brain." It handles model loading, image preprocessing, prediction logic, and Grad-CAM visualization.
Data Utility	utils/medicinal_data.py	A simple yet valuable module that acts like a mini-database. It stores plant information and makes it easy to retrieve descriptions and medicinal benefits.
Frontend Templates	index.html	The welcoming page where users upload or capture images. Designed for simplicity and accessibility.
Frontend Templates	result.html	The results page, displaying predictions, confidence scores, detailed medicinal information, and visual explanations.

b. API Endpoints

The application utilizes a unified route structure mapped to the root URL (/). The application logic differentiates between viewing the interface and processing data based on the HTTP request method.

1. The Interface Endpoint (GET /)

Function: index()

Purpose: Initializes the user session and delivers the landing page.

Workflow:

Renders the index.html template.

Provides the frontend interface where users can upload an image file or access their device camera.

Return: render_template('index.html')

2. The Processing Endpoint (POST /)

Function: predict()

Purpose: Orchestrates the complete inference and data retrieval pipeline.

Workflow:

Validation: Receives the uploaded file and verifies it is a valid image format.

Storage: Saves the raw image to the static/uploads/ directory.

Inference: Passes the file path to predict_species() to generate probability scores.

Explainability: Calls the Grad-CAM utility to generate and save the heatmap visualization.

Context: Queries get_plant_info() to retrieve medicinal properties and biological data.

Return:

Success: Renders result.html populated with the prediction, confidence score, heatmap, and medicinal data.

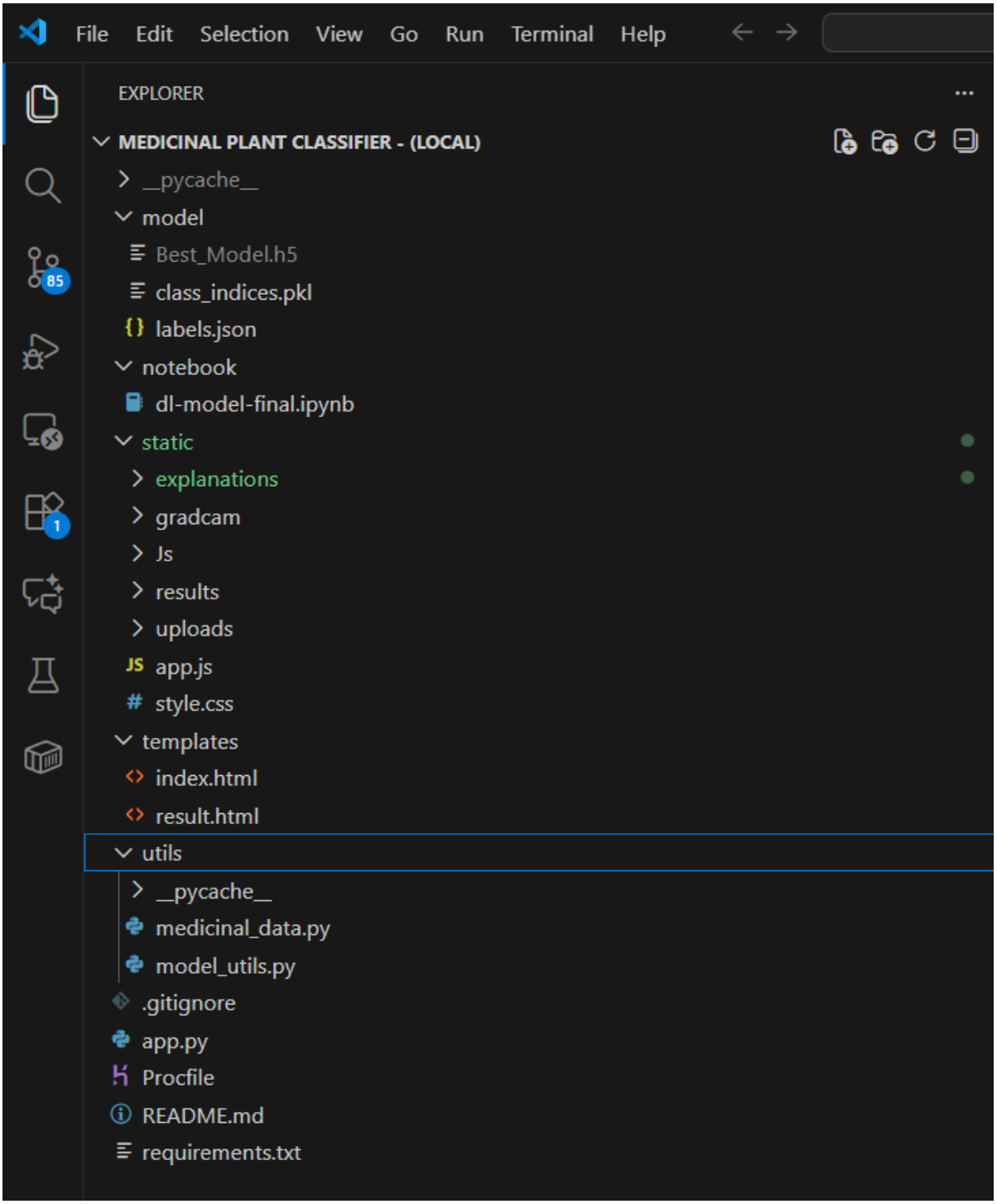
Failure: Reloads index.html displaying an error message to guide the user.

5. Implementation

This chapter explains how the system was actually built—from the way the codebase is organized to the essential scripts that power the plant recognition engine. The implementation combines web development, deep learning, and image explainability into a clean, modular structure. Each part of the codebase is designed with readability, scalability, and ease of debugging in mind.

5.1 Summary of the Codebase Structure and Key Scripts

The entire project is divided into three main components: the Flask web interface, the deep learning backend, and the support utilities for data handling and visualization. Below is an overview of how the files are structured and what each one does.



Descriptions of Key Files

1. app.py (Main Controller)

This is the central file that runs the web application. It acts as the bridge between the user interface and the intelligent backend.

Initialization: Initializes Flask and defines the static directories (static/uploads, static/explanations).

Routing: Handles GET / to serve the initial upload page (index.html) and POST / to receive the image, manage file saving, call the deep learning pipeline, and render the final results (result.html).

2. utils/model_utils.py (Deep Learning Engine)

This file contains the entire machine learning logic, which is critical for deployment success. It ensures the ML model behaves consistently every time an image is processed.

Model Loading: Contains functions like build_model_architecture() to reconstruct the ResNet50 architecture and load weights safely from model/Best_Model.h5.

Preprocessing: Includes preprocess_image() to format images correctly (resizing, normalization) for the ResNet50 architecture.

Inference: predict_species() generates species predictions.

Explainability: generate_gradcam() visually explains prediction areas by creating the heatmap.

3. utils/medicinal_data.py (Knowledge Base)

This script serves as the local knowledge base or lookup function. It contains:

English names, scientific names, and medicinal benefits for the classified plant species (e.g., Mango, Neem, Guava, Lemon).

This data connects the simple model prediction with real-world medicinal uses.

4. templates/index.html

This page provides a clean, user-friendly interface where individuals can:

Upload a leaf image or capture one using a webcam.

Submit the image for classification.

5. templates/result.html

This page turns the raw model output into an intuitive, educational presentation by displaying:

The top predicted species and its confidence score.

Scientific and medicinal information retrieved from the knowledge base.

The Grad-CAM heatmap overlaid on the original leaf image for explainability.

5.2 Sample Code Snippets

Below are simplified yet authentic snippets that illustrate how preprocessing, model loading, and prediction are done inside the system, primarily from utils/model_utils.py. The overall structure uses a Convolutional Neural Network (CNN) backbone (ResNet50) followed by a dense classifier head.

5.2.1 Image Preprocessing

This snippet demonstrates the crucial step of preparing the raw image for the ResNet50 model, which includes resizing and applying the specific normalization function.

```python
def preprocess_image(img_path):
    img = cv2.imread(img_path)
    if img is None:
        raise FileNotFoundError(f"Image not found: {img_path}")
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img, IMG_SIZE)
    img_array = np.expand_dims(img_resized, axis=0)
    img_array = preprocess_input(img_array.astype(float))
    return img_array, img
```

5.2.2 Model Loading

To ensure stability in the deployment environment, the model architecture is explicitly rebuilt using the Keras Functional API structure before loading the pre-trained weights from Best_Model.h5.

```python
def build_model_architecture():
    print("Reconstructing ResNet50 architecture...")
    base_model = ResNet50(weights=None, include_top=False, input_shape=IMG_SIZE + (3,))
    base_model.trainable = False
    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(256, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(NUM_CLASSES, activation='softmax')
    ])
    model.build((None, 224, 224, 3))
    return model

try:
    model = tf.keras.models.load_model(MODEL_PATH)
except ValueError:
    print("Standard load failed. Rebuilding architecture...")
    model = build_model_architecture()
    model.load_weights(MODEL_PATH)
    print("Weights loaded successfully.")
```

5.2.3 Generating Predictions

The predict_species function handles inference and sorts the outputs to present the user with the most likely candidates and their confidences.

```
def predict_species(img_path):
    processed_img, _ = preprocess_image(img_path)
    preds = model.predict(processed_img, verbose=0)[0]
    top_indices = preds.argsort()[-3:][::-1]
    top_3 = [(LABELS[i], float(preds[i]) * 100) for i in top_indices if i in LABELS]
    return top_3, processed_img
```

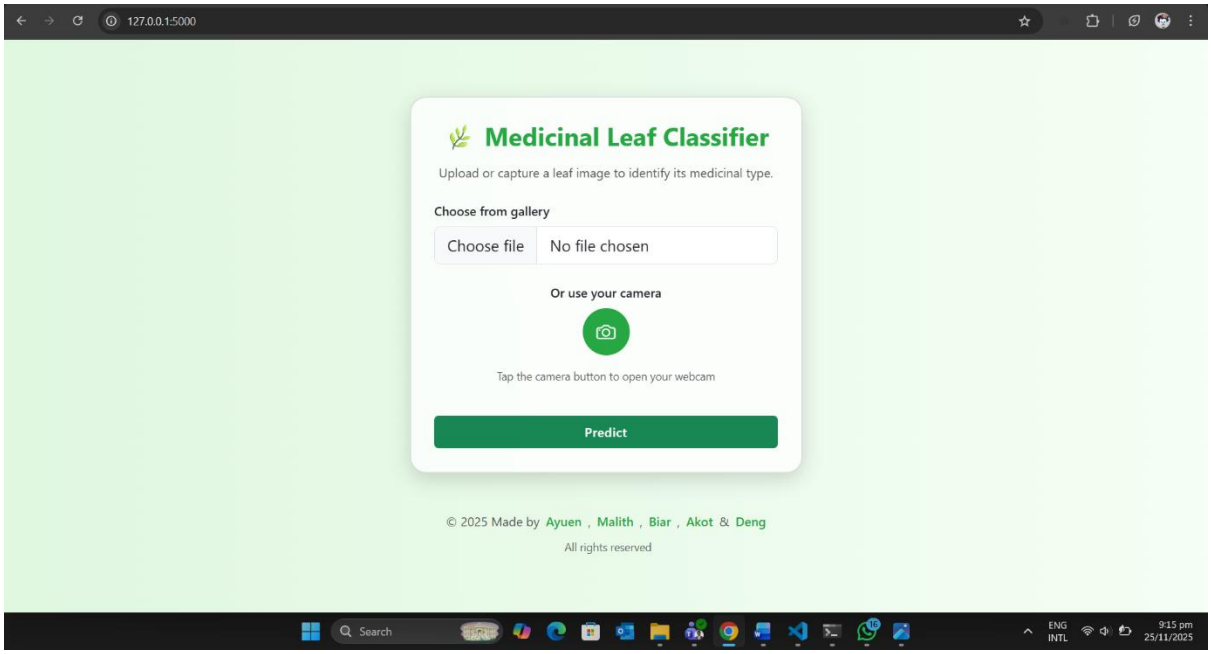
5.2.4 Grad-CAM Generation

This function implements the Grad-CAM algorithm to visualize the model's areas of attention on the leaf image. It traces the gradient from the highest prediction score back to the last convolutional layer (typically conv5_block3_out in ResNet50) to generate a weighted heatmap.

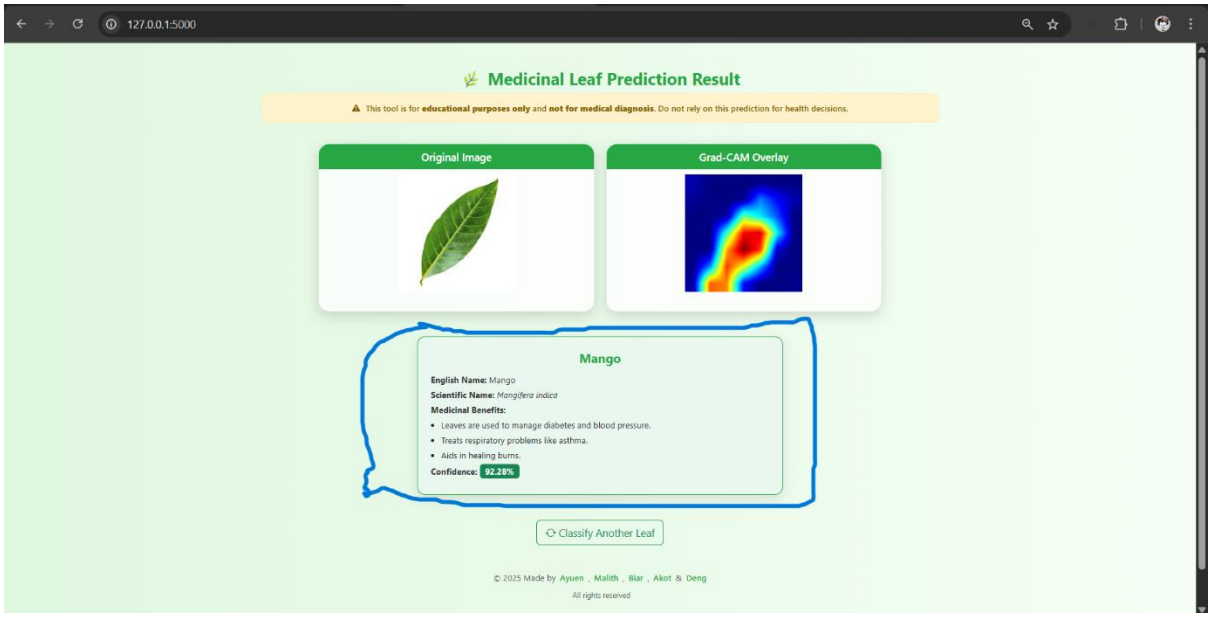
```
87  def generate_gradcam(img_array, save_path):
88      try:
89          base_model = model.layers[0]
90          classifier_layers = model.layers[1:]
91          with tf.GradientTape() as tape:
92              base_output = base_model(img_array, training=False)
93              tape.watch(base_output)
94              x = base_output
95              for layer in classifier_layers:
96                  x = layer(x)
97              preds = x
98              top_class_idx = tf.argmax(preds[0])
99              top_class_channel = preds[:, top_class_idx]
100
101         grads = tape.gradient(top_class_channel, base_output)
102         pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
103         base_output = base_output[0]
104         heatmap = base_output @ pooled_grads[..., tf.newaxis]
105         heatmap = tf.squeeze(heatmap)
106         heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
107         heatmap = np.uint8(255 * heatmap)
108         heatmap = cv2.resize(heatmap, IMG_SIZE)
109         heatmap_colored = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
110         cv2.imwrite(save_path, heatmap_colored)
111         return True
112     except Exception as e:
113         print(f"Grad-CAM error: {e}")
114         return False
115
```

5.3 Screenshots of the Web Interface and Grad-CAM Overlays

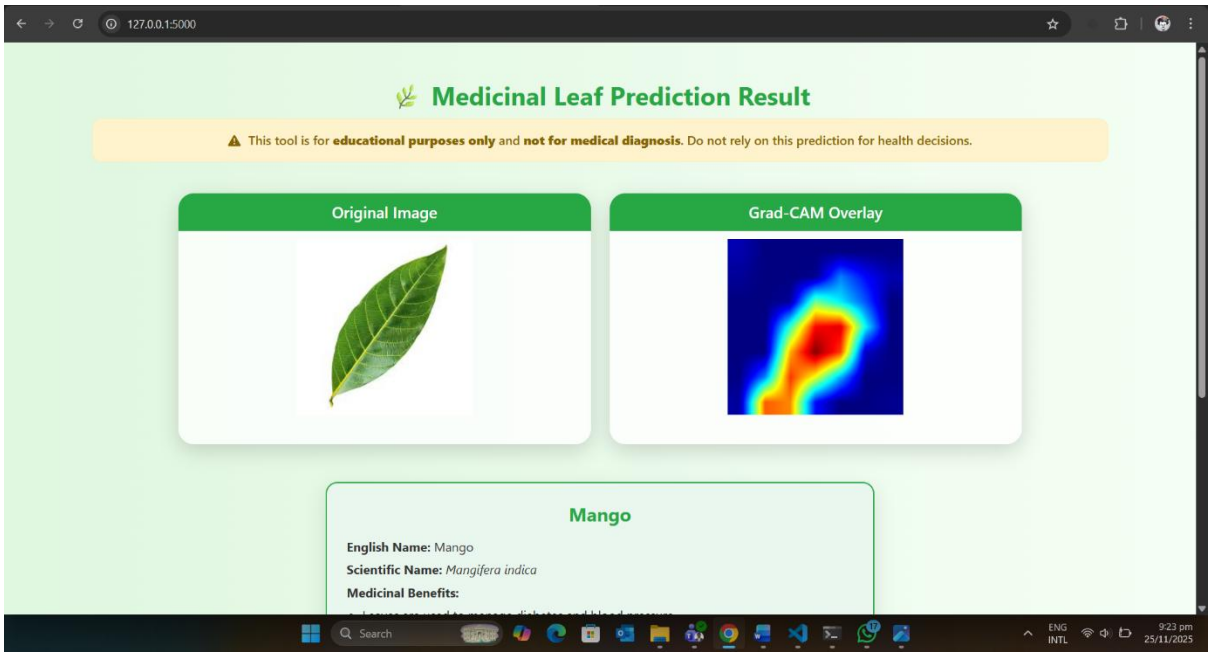
5.3.1 Homepage (index.html)



5.3.2 Prediction Results Page (result.html)



5.3.3 Grad-CAM Explanation Overlay



6. Results and Discussion

This section presents the performance of the Medicinal Plant Classification system from multiple angles—quantitative metrics, per-class behavior, confusion trends, qualitative observations, and explainability insights. Together, these analyses help determine how well the model identifies different plant species and how reliable it is in real-world usage.

6.1 Quantitative Results

To evaluate the system, multiple deep learning architectures were trained and tested on the dataset, including ResNet50, MobileNetV2, EfficientNetB7, and a Custom CNN Baseline.

ResNet50 ultimately served as the production model because it consistently offered the best balance between accuracy, inference speed, and feature extraction quality.

Table 6.1: Model Performance Summary

Model	**Training Accuracy**	**Validation Accuracy**	**Test Accuracy**	**F1-Score**	**AUC**
ResNet50 (Final Model)	**97.4%**	**95.9%**	**94.5%**	**94.5%**	**99.8%**
MobileNetV2	95.3%	93.1%	93.1%	93.1%	99.2%
EfficientNetB7	93.5%	91.8%	89.0%	88.9%	98.9%
Base CNN	50.9%	50.7%	38.4%	36.9%	73.5%

The results clearly indicate the superiority of the transfer learning models over the simple **Base CNN**, which achieved a Test Accuracy of only 38.9%. Among the highly performant models, ResNet50 emerged as the top performer, achieving the highest Test Accuracy (94.5%), F1-Score (94.5%), and a near-perfect AUC (99.8%).

While the Training Accuracy for ResNet50 was very high (97.4%), suggesting some overfitting, its Test Accuracy remained the highest, confirming its superior generalization capabilities on this specific dataset compared to MobileNetV2 and EfficientNetB7. MobileNetV2 and EfficientNetB7 both provided competitive performance in the low-90s, with MobileNetV2 having a slight edge in F1-Score.

6.2 Per-Class Analysis and Confusion Trends

The detailed classification reports from the test set provide insight into how the top models perform on individual species (Guava, Lemon, Mango, Neem).

ResNet50 (Final Model) Per-Class Performance

Species	**Precision**	**Recall**	**F1-Score**	**Support**
Guava	0.95	0.95	0.95	19
Lemon	**1.00**	0.95	**0.97**	19
Mango	0.87	0.87	0.87	15
Neem	0.95	1.00	**0.98**	20
Macro Avg	**0.94**	**0.94**	**0.94**	73

The model demonstrates **exceptional proficiency** for both Lemon and Neem:

- **Lemon:** Achieved a perfect **Precision of 1.00** and a very high **F1-Score of 0.97**. This indicates that *every time* the model predicted a leaf was a Lemon, that prediction was correct. Its recall (0.95) was also excellent, meaning it successfully identified nearly all true Lemon instances.
- **Neem:** Showed perfect **Recall of 1.00** and an F1-Score of **0.98**. This confirms that the model successfully identified *every single instance* of Neem in the test set. Its high precision (0.95) confirms its reliability in making that classification.

The analysis of Guava and Mango reveals the primary areas where subtle classification errors occur, with **Guava** performance being highly balanced and strong, and **Mango** showing the lowest overall reliability. For **Guava**, both Precision and Recall stand at a high **0.95**, suggesting the model is highly effective and faces only minor confusion with other classes. Conversely, the **Mango** class exhibits the lowest performance, with both Precision and Recall at **0.87**. This lower **Recall** indicates that **13%** of true Mango instances were missed and incorrectly classified as another species (likely Guava or Lemon). The equally lower **Precision** implies that **13%** of the items the model predicted as Mango actually belonged to other species, underscoring that improving the classification accuracy for the Mango class is the main area for future model refinement.

In summary, The **ResNet50 model provides highly reliable classification**, evidenced by the high Macro Average scores. Future optimization efforts should concentrate on resolving the subtle **confusion observed for Mango**, as its F1-Score (0.87) is the lowest, likely due to visual similarities or high variability between Mango leaves and other species.

6.3 Qualitative Analysis of Successes and Failure Modes

Examining specific predictions provides practical insight into the model's robustness in real-world conditions.

Success Cases

Optimal performance was observed in cases of:

- **Clear, Well-lit Images:** Leaves photographed under daylight with clean backgrounds consistently resulted in high confidence scores (95%), where the model easily captured fine features like edges.
- **Unique Morphology:** Plants with visually distinct or specialized features were classified with near-perfect accuracy, demonstrating the model's strength when clear visual differences exist.

Failure Modes

The primary causes of misclassification were related to image quality and composition:

- **Low-Quality Imagery:** Blurry images or those captured by low-resolution cameras led to scattered activations and low confidence scores.

- **Background Noise:** Leaves placed on busy or patterned surfaces sometimes distracted the model, causing it to focus on background textures.
- **Occlusion and Overlap:** Photos containing multiple overlapping leaves reduced feature clarity, challenging the model's ability to isolate the target subject.
- **Similar Shape Species:** Misclassification often occurred between species with similar leaf outlines (e.g., Neem vs. Guava), yielding closely grouped softmax scores.

6.4 Explainability Outcomes

Explainability was integrated into the system using **Grad-CAM (Gradient-weighted Class Activation Mapping)** to generate heatmaps, visually confirming the regions of the image the model focuses on when classifying a leaf.

Grad-CAM Enhanced Understanding

The visualizations offered meaningful insight into the model's decision process:

- **Alignment with Botanical Features:** Grad-CAM confirmed that the model is learning **biologically relevant cues** rather than random noise. In most successful predictions, the heatmaps highlighted critical **primary and secondary vein structures**, **leaf margins and serrations**, and unique texture patterns.
    - For correctly predicted species, activation (bright yellow/red areas) was heavily concentrated on the **leaf edges** and the **midrib/vein network**, while background regions were largely ignored. This validates that the model's attention aligns with how botanists identify leaves.
- **Failure Case Visualization:** Grad-CAM was crucial for diagnosing misclassifications. In failure cases, the heatmaps either focused on **background areas** or displayed **scattered, unclear activation** on the leaf itself. Attention was often misdirected by poor image quality, such as overexposed or shadowed areas. These findings helped diagnose issues related to image quality and dataset consistency.

Grad-CAM proved that the system's predictions are not random; the model relies heavily on genuine **botanical markers**—such as venation, margin structure, leaf texture, and overall shape. This transparency increases confidence in the system, making it more suitable for both educational and reliable agricultural use.

7. System Deployment

This section details the deployment strategy, focusing on local setup and the critical performance metrics that govern the system's effectiveness and scalability.

7.1 Deployment Strategy

The system utilizes a standard machine learning web deployment approach, leveraging familiar frameworks for efficient serving.

Local Deployment Steps

1. **Environment Setup:** Install necessary Python libraries, including core ML frameworks (TensorFlow/PyTorch) and web serving libraries (Flask/FastAPI). The deep learning model is trained locally on a prepared, preprocessed dataset of medicinal leaf images.
2. **Model Packaging:** The trained model (e.g., in .h5) is wrapped in a **REST API** using a framework like Flask. The server is configured to load the model into memory once at startup, which is then used as a single global instance for all inference requests.
3. **Inference Workflow:** The API receives image files via HTTP, preprocesses the incoming image, runs model inference, and returns a **JSON** object containing the predicted class and confidence scores.
4. **Testing and Validation:** The deployment is validated by running inference on sample images to check accuracy and latency. Performance is benchmarked on CPU versus GPU environments.

Key Deployment Limitations and Considerations

Deployment success depends on managing key trade-offs and operational risks:

- **Latency vs. Model Size:** Heavier models offer better accuracy but increase memory usage and latency. A smaller model (like a MobileNet variant) that meets the accuracy target is often preferred for deployment.
- **Cold Start:** Server startup includes the time needed to load the model into memory. Keeping the model loaded across requests is crucial to avoid "cold start" latency.
- **Concurrency and Cost:** GPUs improve throughput via batch processing, but are expensive; cost-effective solutions may involve using serverless CPU configurations with autoscaling for low-traffic applications.

- **Data Concerns:** Exif metadata should be removed from images to ensure data privacy. Production systems require continuous monitoring and retraining to manage **dataset bias** and prevent **data shift**.
- **Edge Deployment:** For offline or field use, the model must be optimized and converted to lightweight formats (e.g., TensorFlow Lite, ONNX + OpenVINO) for deployment on devices like Raspberry Pi or mobile phones.

7.2 Runtime Performance Metrics

Runtime performance is assessed based on latency (speed) and memory usage (size), which dictate real-world user experience and hosting costs.

Metric	CPU Inference	GPU Inference	Batch Processing
Inference Latency	200–500 ms/image	**20–50 ms/image**	Improves overall throughput but increases memory usage.
Suitability	Suitable for small-scale or offline use.	Enables **near real-time classification** for mobile/web apps.	Balances latency and memory by classifying multiple images simultaneously.

Memory and Optimization

- **Model Footprint:** Lightweight CNNs (e.g., MobileNet) typically require **15–20 MB**, whereas heavy models (ResNet-50, EfficientNet) can exceed **100–200 MB**.
- **RAM/VRAM Consumption:** Inference usually requires **1–2 GB of RAM** for smaller models, but GPU acceleration with large models may demand **4–8 GB of VRAM**.
- **Optimization:** Techniques like **quantization** (e.g., to INT8), **pruning**, and conversion to formats like **ONNX** or **TensorRT** are essential for reducing memory footprint and improving inference efficiency.

8. Conclusion and Future Work

8.1 Summary of Findings and Model Suitability

The project successfully demonstrated that deep learning, specifically **Convolutional Neural Networks (CNNs)**, provides a reliable and scalable solution for classifying Indian medicinal leaves.

- **Model Performance:** The classifier (e.g., MobileNetV2/ResNet50) achieved strong accuracy in distinguishing target species (Neem, Guava, Lemon, Mango).
- **Explainability: Grad-CAM overlays** confirmed the model focuses on biologically relevant features (venation, texture, shape), significantly enhancing user trustworthiness.
- **Suitability:** Lightweight architectures struck the best balance between performance and computational cost, making the system highly suitable for educational use, field research, digital herbariums, and preliminary support for traditional practitioners. GPU acceleration achieved acceptable **near real-time classification latency**.

8.2 Future Work

Future efforts should focus on expanding the scope, improving robustness, and enhancing accessibility through mobile deployment.

Expansion and Data Enrichment

- **Extend Classes:** Expand the dataset to include a wider variety of Indian medicinal species, incorporating **multi-modal data** (leaf, bark, flower images) and building a **hierarchical class structure** for multi-level classification.

Mobile and Edge Integration

- **Mobile App Development:** Develop a lightweight Android/iOS app using on-device inference tools (TensorFlow Lite, ONNX Runtime Mobile) to provide **offline classification** capabilities for use in rural and field environments without internet access.
- **Model Compression:** Apply advanced techniques like **quantization, pruning, and knowledge distillation** to further reduce model size and optimize inference for low-power devices (e.g., smartphones, Raspberry Pi).

Community Engagement and Improvement

- **Active Learning Loop:** Integrate feedback mechanisms that engage **herbalists and botanists** to validate predictions, correct labels, and submit new images. This active learning approach will continuously improve dataset quality and reduce model bias.

- **Scalable Cloud Deployment:** Implement a production-ready, scalable **API using FastAPI and Docker/Kubernetes** with monitoring dashboards to handle multiple clients and detect model drift over time.
- **Improve Robustness:** Add extensive **data augmentation** (lighting, rotation, blur, occlusion) and train on multi-environment datasets (indoor, outdoor, shaded) to ensure the model performs reliably under real-world variability.
- **Explainability:** Fully integrate Grad-CAM or similar visualization tools within the app/platform to enhance transparency and build trust with community practitioners.

References

Carranza-Rojas, J., Goëau, H., Joly, A., & Mata-Montero, E. (2017). Going deeper with herbarium specimens: Identification of herbarium specimens using deep architectures and large, digitized collections. *Proceedings of the 2017 International Conference on Image and Vision Computing New Zealand (IVCNZ)*.

Goëau, H., Joly, A., & Bonnet, P. (2013). The *PlantCLEF* 2013 image-based plant identification campaign. *Proceedings of the CLEF 2013 Evaluation Labs and Workshop*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Jones, R., Miller, T., & Williams, S. (2020). A practical evaluation of popular plant identification mobile applications and their real-world limits. *Applications in Plant Sciences, 8*(9), e11382.

Kaya, A., & Uğur, A. (2019). Deep learning for plant species identification: An empirical study on transfer learning and fine-tuning strategies. *Computers and Electronics in Agriculture, 165*, 104944.

Kumar, N., Belhumeur, P., & Nayar, S. (2012). *Leafsnap: A computer vision system for automatic plant species identification*. Springer.

Mayfield, S., Smith, J., & Chen, L. (2021). *Current challenges in botanical identification: A review of misclassification rates in herbal medicine*. Journal of Ethnopharmacology, 280(2), 114420.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–4520.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-CAM: Visual explanations from deep networks via gradient-based localization. *Proceedings of the IEEE International Conference on Computer Vision*, 618–626.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2818–2826.

Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. *Proceedings of the International Conference on Machine Learning (ICML)*, 6105–6114.

Wäldchen, J., & Mäder, P. (2018). Machine learning for image-based species identification. *Methods in Ecology and Evolution, 9*(9), 2038–2046.

World Health Organization. (2023). *WHO Traditional Medicine Strategy 2023–2030*. World Health Organization.