Local Medicinal Plant Leaf Classifier Using CNN

Acuña, Kurt Lou Vandrich
Technological University of the Philippines–Manila
College of Science–Computer Studies
kurtlouvandrich.acuna@tup.edu.ph

Delizo, Marlon Kyle
Technological University of the Philippines–Manila
College of Science–Computer Studies
marlonkyle.delizo@tup.edu.ph

Vinuya, Harvey
Technological University of the Philippines–Manila
College of Science–Computer Studies
harvey.vinuya@tup.edu.ph

Abstract—The increasing importance of traditional medicinal plants in healthcare necessitates accurate identification tools to support their conservation and effective use. The Local Medicinal Plant Leaf Classifier addresses this challenge by leveraging Convolutional Neural Networks (CNNs) to classify medicinal plant species based on leaf images. Developed using TensorFlow Keras, the classifier employs a robust dataset augmented with diverse images to improve accuracy and generalization. The model achieved an accuracy of 97.76%, demonstrating its effectiveness in identifying five (5) plant species, including horseradish tree, wild marjoram, and painted nettle.

This system integrates seamlessly into a user-friendly application, allowing users to upload or capture leaf images for real-time identification and information retrieval. While the classifier performs exceptionally well under controlled conditions, testing revealed reduced accuracy with images containing diverse backgrounds, highlighting the need for dataset diversification.

By bridging traditional knowledge with modern technology, the project promotes biodiversity conservation, sustainable healthcare practices, and ecological research, empowering communities and advancing medicinal botany. Future enhancements will focus on improving performance in real-world scenarios and addressing misclassifications among closely related plant species.

Keywords—CNN, Tensorflow Keras, local, medicinal plants, leaves, leaf classifier

I. INTRODUCTION

The rich and varied biodiversity of regional medicinal plants, which provide natural cures for a range of illnesses, is essential to traditional healthcare systems. However, because of the similarities in leaf structure, color, and texture, proper identification of these plants is still difficult, particularly for non-experts. An innovative approach to fill this gap is the Local Medicinal Plant Leaf Classifier, which uses Convolutional Neural Network (CNN). CNN is a deep learning technique, to analyze and identify medicinal plant species from leaf images in a way that is accurate, dependable, and easily accessible [1].

The Local Medicinal Plant Leaf Classifier is built with user-friendly features, the project caters to diverse audiences, from students and researchers to farmers and herbalists, enabling them to make informed decisions about plant usage and conservation. By making traditional knowledge more available to scholars, medical professionals, and local communities, this project seeks to conserve it.

Beyond identification, the Local Medicinal Plant Leaf Classifier encourages knowledge of local medicinal resources and sustainability. In addition to helping to preserve local biodiversity, it also supports efforts in ecological research and sustainable healthcare by bridging the gap between traditional knowledge and contemporary technology. This instrument opens the door for future developments in medicinal botany and conservation by fusing scientific innovation with cultural heritage to empower communities.

II. REVIEW OF RELATED STUDIES

Convolutional Neural Network

A Convolutional Neural Network (CNN) is a specialized type of deep learning model designed to process data with a grid-like topology, such as images. CNNs consist of layers that perform operations like convolution, pooling, and fully connected processing. The convolutional layers use filters (or kernels) to detect features such as edges, corners, or textures from the input data, while pooling layers reduce the spatial dimensions to make computations more efficient and reduce overfitting. Activation functions, such as ReLU (Rectified Linear Unit), introduce non-linearity, allowing the model to learn complex patterns. Fully connected layers take the extracted features and use them for tasks like classification or regression.

CNNs are widely used in computer vision applications, including image recognition, object detection, and medical imaging. For example, they enable tasks like identifying objects in pictures, diagnosing diseases from X-rays, or powering self-driving cars to recognize traffic signs and pedestrians. Their layered architecture mimics the way the human visual cortex processes visual information, making them exceptionally effective for tasks involving visual data. With advancements in frameworks like TensorFlow and PyTorch, CNNs have become accessible and adaptable for diverse real-world applications [1].

Tensorflow

TensorFlow is an open-source machine learning framework developed by Google that allows developers to build, train, and deploy machine learning models. It provides a comprehensive ecosystem for creating both simple and complex neural networks for tasks like image classification, natural language processing, and reinforcement learning. TensorFlow supports a variety of programming languages, with Python being the most widely used. It includes high-level APIs like Keras for easier model building and low-level functionalities for more advanced customization [2].

TensorFlow is highly scalable, enabling it to run on multiple platforms such as CPUs, GPUs, and TPUs (Tensor Processing Units). It offers tools for deployment across devices, from servers to mobile devices and even edge computing environments. Additionally, TensorFlow provides pre-trained models and visualization tools like TensorBoard to monitor training progress. Its flexibility and robustness have made it one of the most popular libraries for machine learning and deep learning tasks in both research and industry [2].

Image Classification

Image Classification is a computer vision task where an algorithm assigns a label or category to an image based on its visual content. It involves training a model to recognize patterns, textures, and features within the image to correctly classify it into predefined categories. For example, an image classification model could distinguish between images of cats, dogs, or cars. This process typically uses deep learning techniques, such as Convolutional Neural Networks (CNNs), to automatically extract features from images and make predictions [3].

The image classification process generally includes several steps: collecting and preprocessing labeled image data, training a model on the data using supervised learning techniques, and evaluating the model's accuracy. Advanced models often use data augmentation and transfer learning with pre-trained models to improve performance. Applications of image classification range from medical diagnosis (e.g., detecting diseases from X-rays) to autonomous vehicles (e.g., recognizing road signs) and e-commerce (e.g., identifying products in images) [3].

III. METHODS

A. Acquisition of the Dataset and Pre-processing

The initial dataset was acquired in Kaggle from PhilMedic. The dataset has five (5) different species of medicinal plant leaves with an average of 100 images per class. Since the number of images from the initial dataset are not enough to train the model to be robust and general, some of the images are acquired through web scraping and selecting high quality images that can be used. To further enhance the performance of the model, a class called ‘unclassified’ is added to the classes where random images like images of people and animals are placed to make sure that the model will only classify leaves of plants and not anything else. After gathering the mixed dataset of the initial images from Kaggle and the web scraped images, we employed a technique called *data augmentation*; it further increased the size and diversity of the final dataset by making variations of data images. The final dataset that is used to train the model comprises a total of twelve thousand, four hundred one (12,401) high quality and curated images.

No. of Classes	Classes	No. of images per class	Split
6	Horseradish tree or Malunggay leaves;	2,351	Training 78.7% Validation 19.9% Testing 1.4%
	Wild marjoram or Oregano leaves;	2,215	
	Painted Nettle or Mayana leaves;	2,045	
	Aloe vera or Sabila leaves;	2,215	
	Sweet potato or Kamote leaves;	1,875	
	Unclassified	1,700	

Table 1. The number and names of the classes with its respective number of images per class, and partition of the images for each of the sets.

B. Tensorflow Keras

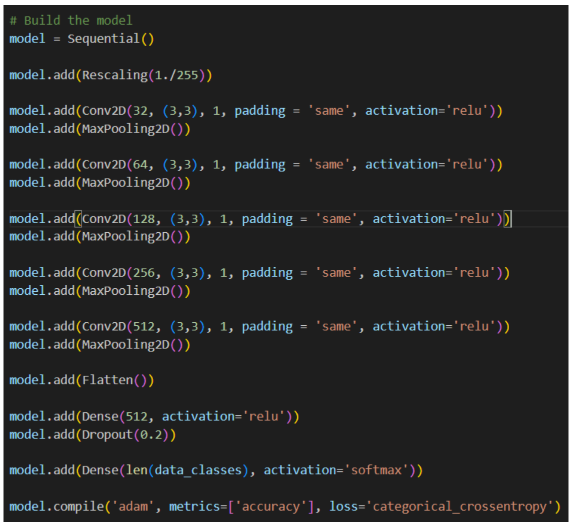
The "Local Medicinal Plant Leaf Classifier" project relies heavily on TensorFlow Keras, which offers a user-friendly and robust framework for creating and refining deep learning models. It enables us to build convolutional neural networks (CNNs), preprocess the leaf image collection, and adjust hyperparameters to maximize efficiency. For feature extraction and classification, Keras makes it easier to create and stack layers, including convolutional, pooling, and fully connected layers. The model is capable of handling intricate tasks like identifying minute patterns in the textures and forms of leaves, which are essential for precisely identifying medicinal plant species, by utilizing TensorFlow's processing efficiency. Furthermore, even with small datasets, Keras makes it possible to easily integrate technologies like data augmentation and transfer learning to improve the classifier's accuracy.

Keras makes it easier to deploy the CNN for practical use after it has been trained. The learned classifier may be included into standalone desktop tools, online apps, or mobile apps because it supports exporting models in formats that work with different platforms. Moreover, TensorFlow Keras offers choices for inference performance optimization, ensuring that the model operates well even on devices with constrained processing power [4]. TensorFlow Keras was the ideal choice to develop the "Local Medicinal Plant Leaf Classifier," with its complete capacity that makes it an efficient instrument for researchers, farmers, and practitioners of traditional medicine.

C. Training

The training of the model consists of several methods to make it generalize well and have a more accurate prediction. The collected images in the dataset undergo a process called *'data augmentation'* to further increase the number of images in the dataset and enhance model generalization. The data augmentation employed are: rotating the images in 45, 90, and 180 degrees, flipping the images both horizontally and vertically, increasing and decreasing both its brightness and contrast, shrink and enlarge the scale of the image, and zooming randomly to the image. After augmenting, the images are then partitioned into a training set which will be used for training the model, validation set which will be used to evaluate the model while training, and testing sets which will be used to test the model and further evaluate it.

The training process started with pre-processing the size of the images for training into 128x128 pixels and having each image in the same size which is a crucial step to ensure the consistency of the images before training. The code has an 'early stopping' function to stop the training if the model is not improving in three (3) epochs. The code will also save the best model during its training. Next is coding the neural networks of the model using Tensorflow Keras.



```python
# Build the model
model = Sequential()

model.add(Rescaling(1./255))

model.add(Conv2D(32, (3,3), 1, padding = 'same', activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(64, (3,3), 1, padding = 'same', activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(128, (3,3), 1, padding = 'same', activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(256, (3,3), 1, padding = 'same', activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(512, (3,3), 1, padding = 'same', activation='relu'))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(len(data_classes), activation='softmax'))

model.compile('adam', metrics=['accuracy'], loss='categorical_crossentropy')
```

Picture 1. *Building the model layers for training.*

In *Picture 1*, the core of the model consists of several convolutional and pooling layers. The first convolutional layer uses 32 filters, each with a kernel size of *(3, 3)* and a stride of 1. The *padding='same'* ensures that the output size matches the input size, while the ReLU activation function introduces non-linearity. A *MaxPooling2D* layer follows this to reduce the spatial dimensions of the feature maps, which helps retain the most important features and minimizes computation. This pattern of a convolutional layer followed by a pooling layer is repeated multiple times, with the number of filters in the convolutional layers increasing progressively (64, 128, 256, and 512). This design enables the model to learn increasingly complex and abstract features as the network deepens [5].

After the convolutional and pooling layers, the model flattens the multi-dimensional feature maps into a *1D vector* using the Flatten layer. This vector is then passed to a dense layer with *512 neurons* and a *ReLU* activation function, which allows the model to learn high-level representations. A dropout layer is added next, randomly setting 20% of the neurons to zero during training to help prevent overfitting and improve generalization [5].

Finally, the output layer uses a dense layer with a number of neurons equal to the number of classes *(len(data_classes))* and a softmax activation function. The softmax function converts the output into probabilities for each class, making it suitable for multi-class classification tasks. The model is compiled with the Adam optimizer, which adapts the learning rate during training for efficient optimization. The loss function is set to *categorical crossentropy*, which is appropriate for multi-class classification problems, and accuracy is used as the evaluation metric [5].

D. System Application

The final model is then applied to a system application using *TKinter*, a Python library and is named 'Local Medicinal Leaf Classifier'.

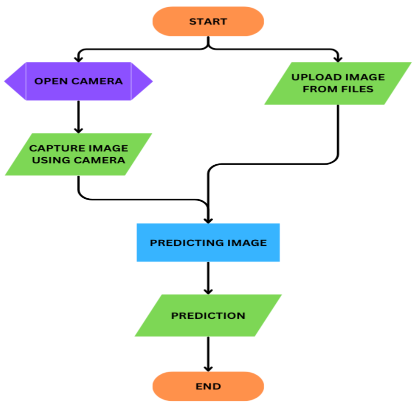
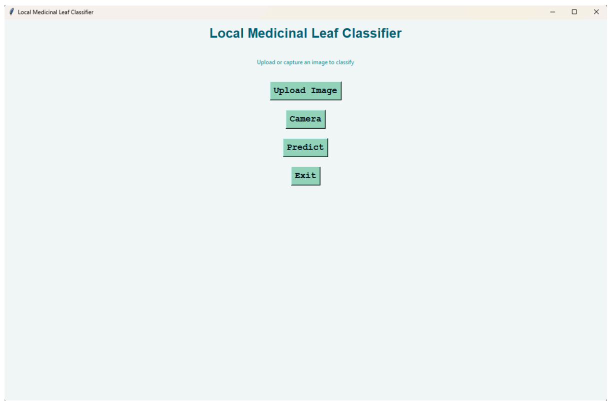


Figure 1. *Flowchart of Local Medicinal Leaf Classifier*

When using the system, the user will be shown the *User Interface (UI)*. The user will choose whether to upload a picture or open their camera and take a live picture to use in the system. If a picture is finally attached to the system, the system will show it to the screen. The user can now click the *'Predict'* button and the system will now process the image and predict the name of the leaf and display information about it [6], if the attached image is one of the classes inside the system. If not, the system will predict it as *'unclassified'*.


Local Medicinal Leaf Classifier
Local Medicinal Leaf Classifier
Upload or capture an image to classify
Upload Image
Camera
Predict
Exit

Picture 2. *User Interface of Local Medicinal Leaf Classifier*

IV. RESULT AND DISCUSSION

After training the model in over ten (10) epochs, its training and validation loss are plotted in a line graph, so that it can be evaluated and see whether the model can be improved. Keep in mind that the code has enabled *'early stopping'* and it saves the best model during its training.

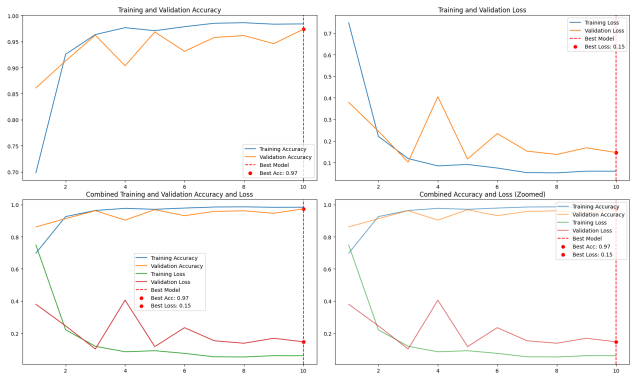


Figure 2. *Four subplots that collectively illustrate the training and validation performance of the model*

In Figure 2, the top-left plot shows the Training and Validation Accuracy, where the training accuracy (blue line) steadily increases and stabilizes near 1.0, indicating that the model successfully fits the training data. The validation accuracy (orange line) follows a similar upward trend but fluctuates slightly, stabilizing at approximately 0.97. A red vertical dashed line marks the epoch where the model achieves its best performance, accompanied by a red dot highlighting the maximum validation accuracy of 0.97. The top-right plot depicts the Training and Validation Loss, where the training loss (blue line) consistently decreases over epochs, indicating effective learning, while the validation loss (orange line) also decreases but exhibits significant fluctuations. Again, the best epoch is marked with a red dashed line and a red dot, representing the lowest validation loss of 0.15.

The bottom-left plot provides a comprehensive view by overlaying all metrics in the Combined Training and Validation Accuracy and Loss graph. Here, the training accuracy (blue) and validation accuracy (orange) trends closely align, with validation accuracy slightly lower, while training loss (green) steadily declines, and validation loss (red) shows variability but generally trends downward. The red dashed line and dot emphasize the epoch where the best combination of low validation loss and high validation accuracy is achieved. The bottom-right plot, labeled Combined Accuracy and Loss (Zoomed), offers a closer look at the same metrics, highlighting the finer details of the fluctuations in validation loss and accuracy. The red markers again pinpoint the epoch where the model performs best according to the selected criteria.

Overall, the model demonstrates effective learning with steadily increasing accuracy and decreasing loss on the training data, but the fluctuating validation loss suggests potential overfitting or variability in the validation dataset. The best model is identified at the marked epoch, with a validation accuracy of 0.97 and a validation loss of 0.15.

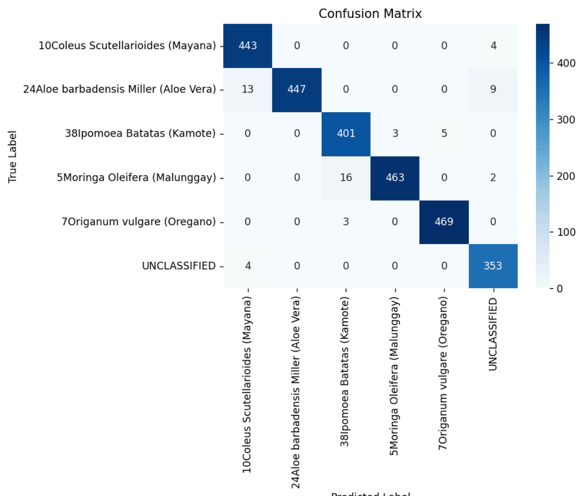


Figure 3. *Multi-class Confusion Matrix of Validation and Test Sets*

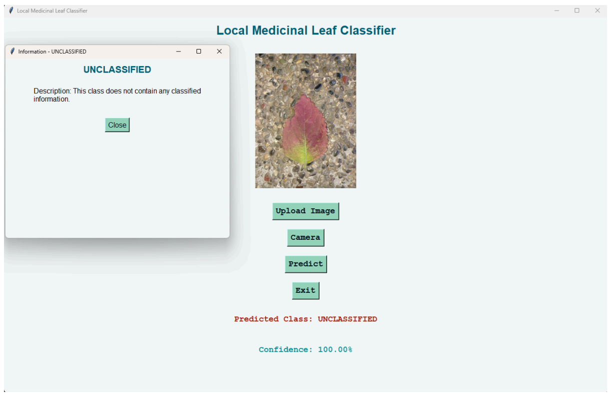
Figure 3 provides a summary of the classification model's performance across six classes. Most predictions lie along the diagonal, indicating a high level of accuracy for correctly classifying samples into their respective categories. For instance, the model correctly classified 443 samples of Coleus Scutellarioides (Mayana), 447 samples of Aloe barbadensis Miller (Aloe Vera), 401 samples of Ipomoea Batatas (Kamote), 463 samples of Moringa Oleifera (Malunggay), 469 samples of Origanum vulgare (Oregano), and 353 samples as Unclassified. However, some misclassifications are evident in the off-diagonal cells. For example, Aloe Vera samples were misclassified as Mayana, Malunggay samples were misclassified as Kamote, and Aloe Vera samples were categorized as Unclassified. Additionally, a few errors occurred between closely related classes, such as Kamote being misclassified as Oregano or Malunggay, suggesting possible overlaps in their distinguishing features. Despite these errors, the model demonstrates strong overall performance.

TP	2,576
FP	23
FN	36
Total Classifications	2,635

Table 2. *Total of True Positives, False Positives, and False Negatives in the Multi-class Confusion Matrix*

$$Accuracy = 2576/2635 = 0.9776 * 100\% = 97.76\%$$

Based on *Table 2*, the accuracy of the model in validation and test sets is 97.76%. This is calculated using the formula for accuracy which is *Accuracy = Total True Positives / Total of Classifications within the Confusion Matrix*.



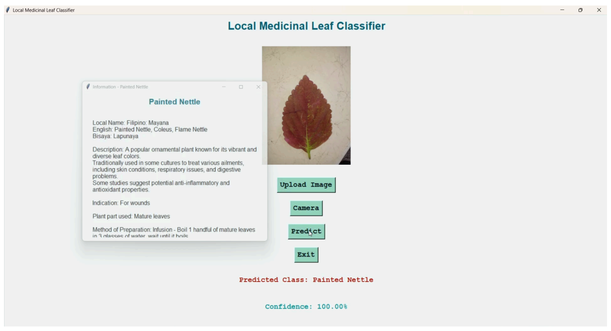
UNCLASSIFIED

Description: This class does not contain any classified information.

Predicted Class: UNCLASSIFIED

Confidence: 100.00%

Picture 3. *The model predicting an image of Painted Nettle or Mayana leaves with diverse background as unclassified*



Painted Nettle

Local Name: Filipino: Mayana
English: Painted Nettle, Coleus, Flame Nettle
Bisaya: Lapunaya

Description: A popular ornamental plant known for its vibrant and diverse leaf colors.
Traditionally used in some cultures to treat various ailments, including skin conditions, respiratory issues, and digestive problems.
Some studies suggest potential anti-inflammatory and antioxidant properties.

Indication: For wounds

Plant part used: Mature leaves

Method of Preparation: Infusion - Boil 1 handful of mature leaves in 3 glasses of water, wait until it boils

Predicted Class: Painted Nettle

Confidence: 100.00%

Picture 4. *The model predicting an image of Painted Nettle or Mayana leaves with white background as Painted Nettle*

In *Picture 3 and 4*, further analysis reveals a limitation regarding the background of input images. Images with white backgrounds are classified more accurately, aligning with the characteristics of the training dataset. However, when images include complex or non-white backgrounds, the model's prediction accuracy declines. This indicates that the model's performance is biased toward the conditions of the training data and suggests a need for inclusion of diverse backgrounds in the dataset for more robust predictions in real-world scenarios.

V. CONCLUSION

In conclusion, the *Local Medicinal Leaf Classifier* demonstrates highly effective learning and classification capabilities, achieving near-perfect accuracy in controlled conditions. To enhance its real-world applicability, future work should focus on expanding the diversity of the training dataset, particularly in terms of background variations, and exploring techniques to minimize misclassifications between closely related classes.

VI. REFERENCES

[1] Gori, M., Precioso, F., & Trentin, E. (2023). Deep learning. In *Cambridge University Press eBooks* (pp. 301–349). https://doi.org/10.1017/9781108755610.012
[2] *Introduction to TensorFlow*. (n.d.). TensorFlow. https://www.tensorflow.org/learn
[3] *What is image classification? Basics you need to know | SuperAnnotate*. (n.d.). SuperAnnotate. https://www.superannotate.com/blog/image-classification-basics
[4] *Keras: The high-level API for TensorFlow*. (n.d.). TensorFlow. https://www.tensorflow.org/guide/keras
[5] Team, K. (n.d.). *Keras documentation: Keras 3 API documentation*. https://keras.io/api/
[6] *Plant compendium - Philippine Traditional Knowledge Digital Library on Health*. (n.d.). https://www.tkdlph.com/index.php/ct-menu-item-3/ct-menu-item-13