Group Machine Learning Project

Course Name: Machine Learning - COSC2753
Group: S2_G1
Team Member:
Nguyen Khanh An - s4019548
Hong Thieu Kiet - s3993986
Ung Xuan Dat - s3932156
Pham Quang Huy - s3970891

1. Introduction

In this project, we will walk you through our approach to build a solution for predicting various aspects of the rice plant, including disease, variety and age. These solutions will come as three separate models, with each one responsible for predicting each attribute of the rice plant, and there will be a website acting as the interface between the farmer and these models to make our solution accessible to the majority of our intended audience. This project aims to aid farmers in effective crop management by giving them the ability to identify these attributes as conveniently and accurately as possible, and thus mitigating common issues such as crop losses.

The project aims to tackle these three task:

- **Task 1**: Identifying the disease afflicting the rice paddy. This task aims to categorise the rice paddy based on a set of given diseases, so it is a classification task.
- **Task 2**: Identifying the paddy variety. This task aims to categorise the rice paddy based on paddy variety based on a set of given variety, so it is a classification task.
- **Task 3**: Identifying the rice paddy age. This task aims to predict the rice paddy's age based on a range of numeric values, so it is a regression task.

2. Exploratory Data Analysis (EDA)

As a given when doing any end-to-end project, the first thing to do is analyse the provided data by performing the EDA process on it. Here are the insights that we gathered:

- **Given dataset:** The dataset that was given consisted of 10,407 images, which given that we are going to train models for recognising images, is barely enough to train a small model. Along that is a metadata file that labels all of these images, represented as filename in this file, with diseases, varieties and ages. There is also the test dataset which consists of 3469 images which are unlabeled. This is meant for us to run through our models to give out predictions of all three attributes.
- **Attribute**:
  - **Label (Disease)**: A categorical attribute that contains 10 unique values. The label with the highest count is "blast" with 1764 instances and the lowest one is "bacterial_panicle_blight" with 337 instances, and the other eight lie around that range. Although these numbers suggest some class imbalance, it should not hinder training significantly.
  - **Variety**: A categorical attribute that contains 10 unique values. This attribute contains an extreme imbalance, having 6992 "ADT45" labels on the highest end and only 32 "Surya" labels on the lowest end. This is a very significant imbalance that may negatively impact model performance, as models tend to be biased toward the majority class.
  - **Age**: A numerical attribute that ranges from 45 to 82. This attribute distribution follows a multimodal pattern, meaning that it has multiple peaks rather than a single central peak. It also appears to be skewed toward the 70s mark.
- **Image dimension**: Most of the images featured in the dataset follow the size of 640 x 480, with few having a size of 480 x 640 instead.
- **Image file format**: All of them are in JPG format, which is supported by all of the API that was used during this project.
- **Data Duplication**: There are no duplicate entries found.
- **Missing Label**: There are no entries with missing labels.

3. Data Preprocessing

In order to properly train the model, we first have to preprocess the image data into a suitable format for our models to read and train from it.

- **Loading the image from the directory**: Because our project involves preparing three models that perform different tasks, that means when it comes to training, we need to have three separate data pipelines for each of the tasks, since they need to contain different labels for each task. We are not going to load three copies of the dataset into the memory because that would not be computationally efficient. Instead, we will define three dataset using **tf.data.Dataset**, which will read images from disk lazily, which allows us to have three separate datasets configuration that come from the same data source. Additionally, when the images are loaded, it will read them as 224 x 224 images instead of 600 x 400. This is because for all of the models featured here, they are all expecting images that fall into a specific size.

- **Splitting the data**: In order to train any model, there needs to be the train dataset and the validation dataset. The split ratio that we have opted for is an 80/20 split.
- **Image Normalization**: In any neural network or deep learning project that deals with images, normalization is an essential step to apply to the data before commencing training. Image normalization refers to the technique of scaling pixel values so that it falls within a specific range or distribution, in this case the image will be scaled from 0-255 to 0-1. This helps to remove scale bias, improve training speed, stabilize gradients and makes training more consistent across batches and datasets. This process will be applied through making it a layer that can be applied at the top of each model. This approach prevents us from manually applying normalization every time we load our data by having the images be normalized when it is being fed to train models.
- **Data Augmentation**: A common problem deep learning models usually run into is overfitting, and it is especially a concern for our particular situation. The fact is that our dataset comprises only around 10000 images and for some of the task's labels, it is heavily underrepresented, resulting in class imbalance. One technique commonly used to mitigate this shortcoming is data augmentation. This technique is used for synthetically creating new images by augmenting existing images by various attributes such as slight rotation, shifting in either the x-axis or the y-axis, zooming or brightness adjustment. Although the augmented samples are derived from the same original images, the variations help the model generalize better and reduce overfitting. The augmentation process, similar to the normalization process, will be a part of the models themselves as a layer placed on top. This approach ensures that each training epoch sees slightly different versions of the data, providing a virtually infinite stream of training samples without increasing the actual dataset size.

4. Evaluation Framework

For our method of evaluation, given that the end product is a web-based application intended for use by farmers, we have adopted a multi-faceted evaluation approach. This includes:

- **Accuracy**: Used as a baseline metric for classification tasks, accuracy reflects the proportion of correct predictions. It helps assess overall model performance during training and fitting, especially when class distributions are relatively balanced.
- **F1 score**: For classification tasks, we include the F1 score to address potential class imbalance and to ensure robust performance across all categories. As the harmonic means of precision and recall, it provides a more balanced view of a model's effectiveness, particularly when false positives and false negatives carry different consequences.
- **Mean Absolute Error (MAE)**: Applied to the regression task (Task 3: predicting plant age), MAE quantifies the average absolute difference between predicted and actual values. It serves as a key metric during model fitting and evaluation, providing an intuitive measure of prediction accuracy.
- **R^2 Score (Coefficient of Determination)**: Used alongside MAE for the regression task, the R^2 score measures the proportion of variance in the target variable that is explained by the model. It offers insight into how well the model captures underlying patterns in the data.
- **Inference Latency**: To ensure responsiveness in real-world use, especially in rural areas with limited internet connectivity, we measure the time taken for the model to return predictions. Low latency is critical for user experience in a browser-based environment.
- **Model Size**: Although models are hosted on the backend, size remains important for deployment efficiency. Smaller models require less memory and storage, reduce loading times on the server, and allow for faster scaling and resource management—especially important when serving multiple users concurrently.

5. Model Selection

Before we get to the main part of this project, the section where we develop the solution through various experimentation and fine-tuning, we have to choose the baseline models that we are going to use.

The first thing we decided on is the type of architecture that we are going to utilize moving forward. Considering the tasks given were all related to label prediction, there is really only one standout option for this, and that is Convoluted Neural Network (CNN). CNNs are specifically designed for image processing, capturing spatial features like edges and textures through convolutional layers. In contrast, traditional fully connected neural networks treat each pixel independently, making them less efficient and prone to overfitting on image data. Given our dataset and objectives, CNNs offer a more accurate and computationally efficient solution.

The next step is to choose the type of models within the CNN categories that act as the baseline for us to train. Based on our research and considering the size of our dataset, we shortlist three potential models for us to experiment. All of the models listed here can be retrieved through Keras Applications, saving us the trouble of building the model ourselves. Note that when we use the model from Keras, we deliberately do not include the weight given to the model as we prefer training these models from scratch.

- **MobileNet (V1)**: MobileNet is a streamlined model designed to be lightweight and fast, making it a premiere option for deployment of our web application. It is based on depth-wise separable convolutions, which significantly reduce the number of parameters and computational cost. Its efficiency makes it well-suited for our dataset of 10,000 images, offering a good balance between performance and resource usage without the need for large-scale data or heavy infrastructure. It is the lightest and fastest model we will be evaluating in this project.
- **EfficientNetB0**: EfficientNetB0 is a compact yet powerful convolutional neural network that achieves high accuracy while maintaining low computational cost. It uses a compound scaling method that uniformly scales depth, width, and resolution, resulting in a well-balanced architecture. It offers an excellent trade-off between performance and efficiency, delivering strong generalization without requiring excessive training time or memory. It is the second lightest and second fastest model we will be evaluating in this project.
- **Xception**: Xception is a deep convolutional neural network that builds upon the Inception architecture by replacing its modules with depthwise separable convolutions. This design allows the model to learn spatial and channel-wise features more efficiently, leading to improved performance with fewer parameters. It is a strong candidate due to its ability to capture complex patterns in image data while maintaining relatively efficient computation. It is the heaviest and slowest model in this project.

For training purposes, we cannot take the model retrieved from Keras on its own, we have to build a custom model pipeline that includes preprocessing layers, a base feature extractor, and additional dense layers for classification or regression. This allows us to tailor the model to our specific dataset by applying data augmentation and normalization, fine-tuning the model for our task, and customizing the classifier head to match the number of output classes. (We will be using the Adam optimizer when we compile the model, as compared to SGD, it is more flexible when it comes to tuning hyper-parameters, which is what we will be discussing in the next section.)

6. Training Strategy

After careful consideration, we have determined the most optimal approach to experiment all three models with all three tasks given the dataset and the timeframe that we were on is for each task, we are going to tune the hyper-parameter of each selected model individually using the validation set, ensuring that we maximize performance for each task. Specifically, we will apply random search for hyperparameter optimization. This approach allows us to tailor the parameters for each model in each task, allowing us to compare the three models at its best.

The parameter we will be tuning are:

- **dense_units**: Number of units in the dense (fully connected) layer. It controls the model's capacity to learn from extracted features.
- **dropout_rate**: Dropout rate for regularization. It helps prevent overfitting by randomly deactivating a fraction of neurons during training.

- **learning_rate**: Learning rate for the Adam optimizer. Learning rate affects how fast the model updates weights. The range is from 1e-4 to 1e-2 (sampled logarithmically).

The tuning process will also be employing other techniques such as early stopping and reducing learning rate on plateau to prevent overfitting and improve training efficiency.

7. Alternate Strategy Considered

In addition to the strategy mentioned above, we also perform experimentation with several techniques in addition to data augmentation in order to address the issues of imbalance and insufficient dataset, particularly prevalent in task 2.

- **GAN**: Used to generate synthetic images to augment minority classes, but we decided not to go that route due to the risk of it generating poor quality image, polluting the training dataset and harming the training process.
- **Class weight**: Adjusts the loss function to penalize misclassifying minority classes more heavily, primarily experimented with task 2, since task 1's data distribution deemed it unnecessary to apply this technique. During implementation, it resulted in extreme performance drop due to the model overcompensating for the minority classes, leading to significant misclassification of the majority class and overall instability in training.
- **Sample weight**: Allows the model to prioritize certain data points by assigning them higher weights in the loss calculation, applied to emphasize samples from underrepresented age groups in task 3. However, it led to unstable training and degraded performance, likely because the model overfit the weighted samples, reducing its ability to generalize across the full age distribution.
- **Oversampling**: Duplicates samples from minority classes to balance the dataset. This led to overfitting, as the model memorized repeated images without learning generalized features.
- **Undersampling**: Removes samples from majority classes to balance the dataset. This reduced the overall training data significantly, causing a drop in model accuracy due to loss of valuable information, especially when dealing with a limited amount of images.

8. Results

Here are the results we have collected after going through the process of tuning a total of 9 models across 3 tasks.

a. Task 1

All 3 models perform similarly well in the validation accuracy front, with EfficientNetB0 and Xception being slightly equally better than MobileNet. In the F1 front, MobileNet under-performs when compared to the other two models, as they achieved roughly the same score. Overall, all three models performance was better than our expectation entailed.

b. Task 2

MobileNet under-performs in the validation accuracy front when compared to the other two, with EfficientNetB0 and Xception being quite equally better than MobileNet. In the F1 front, EfficientNetB0 and MobileNetV1 perform similarly well, but are worse than Xception by quite a distance.

c. Task 3

In terms of MAE, Xception proved to be the best, followed closely by MobileNet and EfficientNetB0 is in last by a good distance. In the R2 score department, we see that EfficientNetB0 has fallen behind by quite a margin, MobileNet is considerably better than EfficientNetB0 but ultimately fell short of the score of Xception.

9. Ultimate Judgement

After a thorough examination of the collected results, we have concluded the best model for each task, plus the limitations we have recognised and steps we should take in the future to augment these.

For task 1, we decided that the best model for this task is EfficientNetB0. This is due to its impressive performance in the F1 score, proving that it can accurately balance precision and recall across the different disease classes better than MobileNet and rivalling Xception. But ultimately, its lower size and latency when compared to Xception make EfficientNetB0 our premiere option for task 1. Through research on papers done on this exact topic, which was plentiful due to predicting paddy disease being a fairly popular subject within the deep learning community, with 2,675 entrants joining the Kaggle competition on this topic [1], it turns out that both EfficientNet and Xception has proven to be premiere choices when it comes to tackling this problem. We cannot definitively compare performance against most of the published solutions since the result they collected is through testing with the test dataset, meanwhile most of our data collected is just a rough estimation of the performance done through the validation dataset due to lack of test label access. But from what we can observe, it seems that our trained models seem to rival or even surpass some of them, as we can see by comparing the result of one of the papers using EfficientNetB0 as one of a potential solution, having its result as 96% compared to our 98.7% [2]. Due to the topic prevalence, there will be an abundance of data for further training and multiple papers to take inspiration from to improve our strategy.

For task 2, we decided that the best model for this task is Xception. Just like in task 1, Xception and EfficientNetB0 perform well, accuracy wise. But unlike task 1, Xception got a better F1 score than EfficientNetB0, and quite a considerable margin too. Considering the imbalanced nature of the task 2 dataset, we decided that the extra performance on Xception is worth it, despite the advantages that EfficientNetB0 held over it, mainly latency and size. Unlike task 1, there is significantly less attention put on predicting the paddy variety. Due to that, the task suffered from a lack of labeled data. Coupled with the issues found in the dataset, the distribution and the lack of entries for some classes, this model will encounter a lot of hardship. That is why we are considering applying the technique of semi-supervised learning in the future, as there is an abundance of labeled data for disease identification. We could use that data to apply this technique to further mitigate this problem.

For task 3, we decided that the best model for this task is MobileNet. Even though scorewise, Xception outperformed it. The fact is that MobileNet has achieved a result rivalling Xception while being twice as fast in prediction speed and 5 times smaller in size. It is significant due to the target demographic being farmers, who do not usually have access to a high-speed internet connection. For that reason, task 3 will utilize MobileNet. Similar to task 2, there is not much labeled dataset for paddy age, resulting in less papers to take resources from. Luckily, the distribution of the dataset is relatively normal, so it is not as bad of an issue as in task 2, but it is an issue nonetheless. Also as in task 2, there are plans to improve the model through the means of semi-supervised learning.

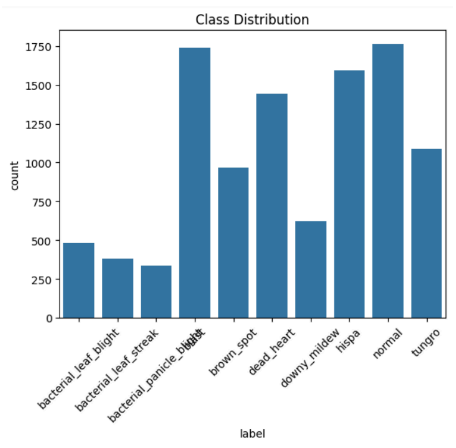
10. **Web Deployment**:

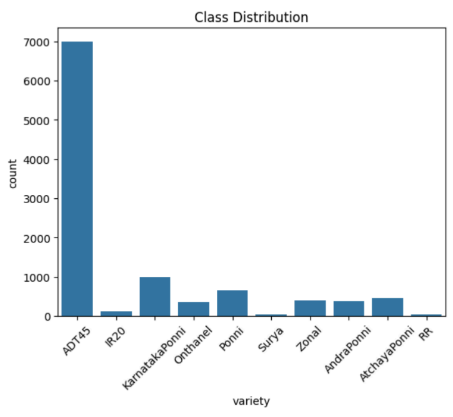
This project is a crop health prediction system that combines a Flask back-end and a React front-end to provide intelligent insights into crop images uploaded by users. The system supports three prediction tasks: identifying the crop variety, detecting potential diseases, and estimating the age of the crop. On the back-end, three deep learning models (Xception, EfficientNetB0, and MobileNet) are used for each respective task. Images uploaded through the interface are securely handled, preprocessed, and passed to the models, which return predictions in JSON format. The front-end offers a clean and responsive interface using Bootstrap, allowing users to select a prediction type, upload an image, preview it, and view results clearly. This tool is designed to support precision agriculture by enabling fast, AI-powered analysis of crop conditions based on visual data.

For future deployment, the project aims to incorporate several enhancements to ensure scalability, reliability, and accessibility. Mobile optimization and offline access will be achieved by implementing Progressive Web App (PWA) features, making the system more usable on low-end devices and under poor network conditions. The back-end will be migrated to a scalable cloud infrastructure to improve performance and ensure reliability during high traffic. Enhanced security measures such as authentication, rate limiting, and caching will be integrated to protect APIs and optimize data access. To reach a broader audience, the platform will support multiple languages through localization. Additionally, automation and monitoring will be implemented by setting up a CI/CD pipeline for streamlined deployment and integrating error tracking tools for real-time issue detection and maintenance.

11. Appendix

a. Train Dataset Labels Distribution





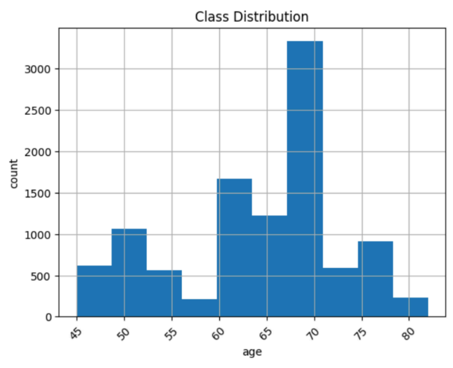
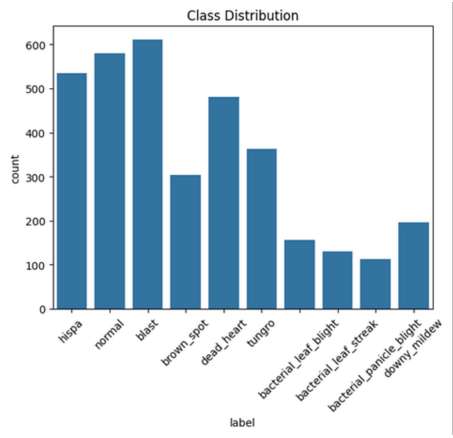


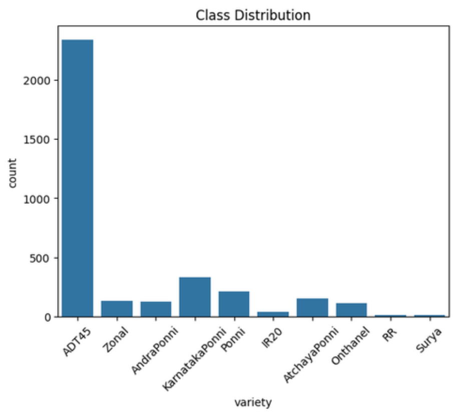
Figure 1: Label (Diseases) distribution

Figure 2: Variety distribution

Figure 3: Age distribution

b. Test Dataset Labels Distribution





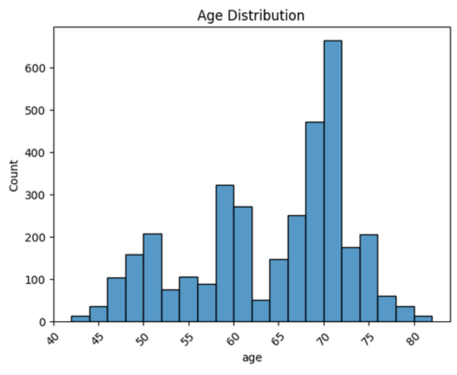


Figure 4: Label (Diseases) test result distribution

Figure 5: Label (Diseases) test result distribution

Figure 6: Label (Diseases) test result distribution

c. Training Results

Task 1	**Validation accuracy**	**F1 score**	**Size (MB)**	**Latency (ms)**
MobileNet (V1)	0.979	0.985	16	1.67
EfficientNetB0	0.987	0.986	29	2.11
Xception	0.987	0.987	88	2.91

Table 1: Task 1 result

Task 2	**Validation accuracy**	**F1 score**	**Size (MB)**	**Latency (ms)**

MobileNet (V1)	0.979	0.977	16	1.93
EfficientNetB0	0.987	0.979	29	2.63
Xception	0.987	0.986	88	3.48

Table 2: Task 1 result

Task 3	**MAE Score**	**R2 score**	**Size (MB)**	**Latency (ms)**
MobileNet (V1)	1.507	0.930	16	1.62
EfficientNetB0	1.851	0.902	29	2.12
Xception	1.310	0.943	88	2.83

Table 3: Task 3 result

12. Citation

[1] "Paddy Doctor: Paddy Disease Classification," *@kaggle*, 2025. https://www.kaggle.com/competitions/paddy-disease-classification.

[2] V. Garg, S. Agarwal and S. Sharma, "Deep Learning-based Paddy Doctor for Sustainable Agriculture," 2023 Seventh International Conference on Image Information Processing (ICIIP), Solan, India, 2023, pp. 485-490, doi: 10.1109/ICIIP61524.2023.10537776. keywords: {Plant diseases;Plants (biology);Transforms;Pesticides;Information processing;Feature extraction;Agriculture;Sustainable agriculture;Convolutional neural networks;Rice disease classification}