

# Grapevine Leaves Image Classification Project

## Introduction

This project aims to develop a machine learning model for classifying grapevine leaves images using deep learning techniques. The dataset for this project is sourced from Kaggle's "Grapevine Leaves Image Dataset." The project is divided into three sections, each focusing on different aspects of machine learning workflows, including data preprocessing, model building, and comparison of models with and without pre-trained weights.

## Dataset Source

- Kaggle: [Grapevine Leaves Image Dataset](#)

## Preparatory Instructional Materials

Before starting the project, participants must watch specific videos and read articles to understand the methodologies and tools involved:

- For Sections 1 and 2:
  - [Video Tutorial](#)
- For Sections 2 and 3:
  - [CNN Multiclass Image Classification Python / Confusion Matrix](#)
  - [Precision, Recall, F1 score, True Positive](#)
  - [Article on Evaluation Metrics for Classification Models](#)

- [VGG16 Transfer Learning](#) (for your implementation, you have to use `layers.trainable=True`)

Note: For computational resources, participants can use their own PC's GPU or cloud platforms like Google Colab or Kaggle. In Google Colab, select the GPU as the runtime type for enhanced performance.

## Section 1: Data Preprocessing

Objectives:

- Download and upload the dataset to Google Drive.
- Mount Google Drive in the Colab environment.
- Load the dataset into the notebook.
- Resize images to 224x224x3, convert them into numpy arrays, normalize by dividing by 255.0, and add them to a list named `images`.
- Create labels for the dataset and use encoders to format them appropriately.
- Plot a bar graph to visualize the distribution of samples across different classes.
- Split the dataset into training (80%) and testing (20%) sets, using stratification.

References:

- Data Splitting and Stratification: [Stratify in Train/Test Split](#)

## Section 2: Building a Custom VGG19 Model

Objectives:

- Construct a custom VGG19 model without using pre-trained weights.
- Compile the model using the Adam optimizer and an appropriate loss function.
- Fit the model to the dataset for 10 epochs, incorporating early stopping and CSVLogger for bonus points.
- Save the trained model.
- Evaluate and display training and testing accuracy.

- Plot graphs for training and testing accuracy and loss, and analyze overfitting or underfitting.
- Generate a classification report and a confusion matrix using the Seaborn library.

References:

- Early Stopping and CSVLogger: [Guide to Early Stopping and CSVLogger](#)

## Section 3: Implementing Pretrained VGG19 Model

Objectives:

- Create a VGG19 architecture with ImageNet pre-trained weights, ensuring all layers are trainable.
- Compile and train the model for 10 epochs.
- Save the trained model.
- Evaluate and display training and testing accuracy.
- Plot graphs for training and testing accuracy and loss, and analyze overfitting or underfitting.
- Generate a classification report and a confusion matrix.

References:

- VGG19 Pretrained Model: [Keras VGG19 Model](#)

## Comparison and Conclusion

Tasks:

- Compare the performance of the models from Sections 2 and 3.
- Discuss which model performed better and why.
- Suggest methods for increasing model accuracy and potential future works on the dataset.