Rice Image Classification using Classical Machine Learning Models

Egor Bykov

May 2025

1 Introduction

In this project, the goal is to develop a machine learning model that can accurately classify rice grains into different types based on images. The dataset contains images of five different rice varieties. The task involves training several machine learning models to distinguish between these rice varieties by extracting relevant features from the images.

Given the high-dimensional nature of image data, preprocessing steps such as dimensionality reduction are necessary to improve model performance and reduce computational complexity. In this project, Principal Component Analysis (PCA) is employed to reduce the number of features while retaining a high percentage of the variance in the data. After PCA, several machine learning classifiers are applied, including Random Forest, K-Nearest Neighbors (KNN), and Logistic Regression.

The project is structured into several key steps:

- Download and preprocess the dataset of rice images.
- Apply dimensionality reduction using PCA.
- Train and evaluate multiple machine learning models.
- Analyze and interpret the results, including accuracy and confusion matrices.

The rice dataset and trained models can be found on Google Disk. The code was implemented in Google Colab. By the end of this project, we aim to identify the most effective model for classifying rice grains.

2 Dataset Description

The dataset used in this project is the **Rice Image Dataset**, which contains color images of five different rice grain varieties:

- Arborio
- Basmati
- Ipsala
- Jasmine
- Karacadag

Each variety is stored in a separate subdirectory, and the images are captured under consistent lighting and scale conditions. The images are originally in JPEG format and have resolution 250×250 pixels. Later they were resized to 32×32 pixels in order to reduce computational cost.

The dataset is organized as follows:

- Each subfolder corresponds to a rice variety and contains 15 000 images.
- Images are named arbitrarily and are assumed to be correctly labeled based on their directory structure.
- The total dataset includes 75 000 images: 15 000 from each class.

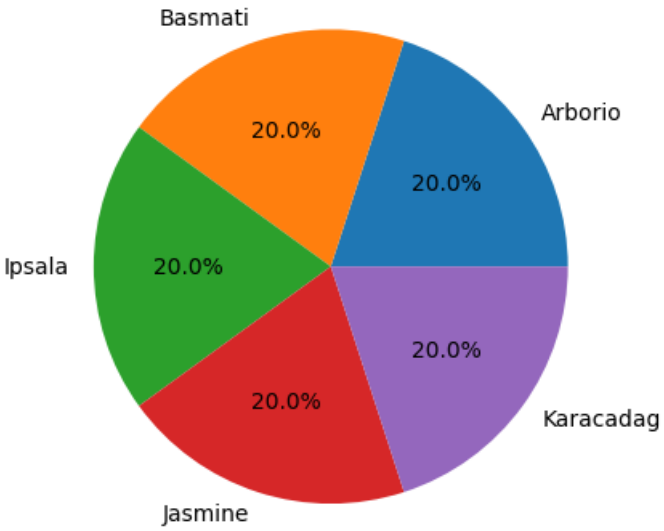


Figure 1: Rice class distribution

Some rice varieties look similar to each other, while other can be easily distinguished. For example, Basmati is similar to Jasmine variety because of its

streched shape, while Arborio and Karacadag are almost round as can be seen in Figure 2. This observation suggests that classification models may misclassify these pairs of rice varieties.

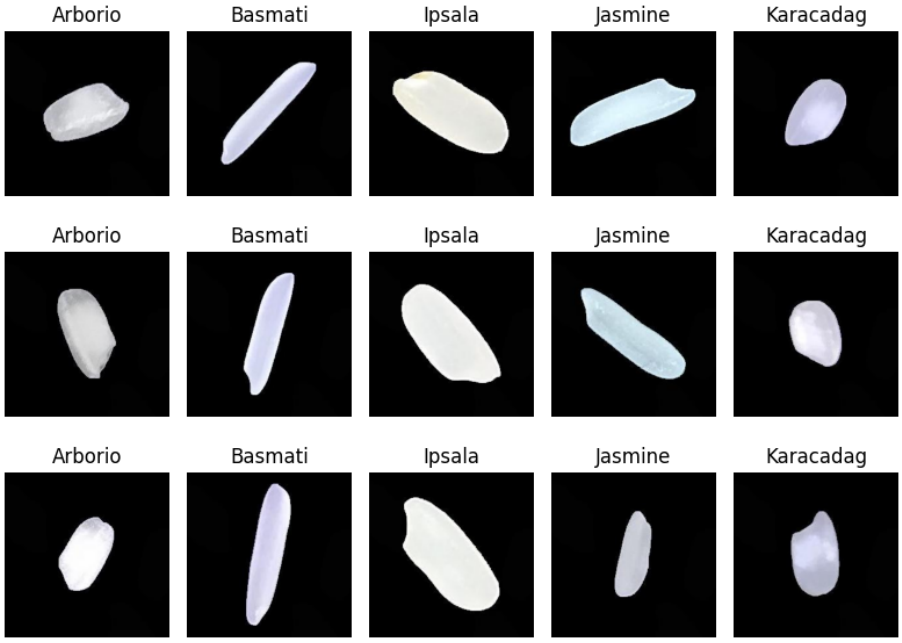


Figure 2: Samples from the dataset

To speed up future computations, the preprocessed image data (after resizing) and the corresponding labels are stored in binary files (`x.pkl` and `y.pkl`) using Python's `pickle` module. This allows quick reloading without the need to repeat the image loading and preprocessing steps.

Each image is represented by a 3-dimensional array of shape $32 \times 32 \times 3$ after resizing, where the last dimension corresponds to RGB color channels. These arrays are then flattened to 1D vectors of length $32 * 32 * 3 = 3072$ before standardization and PCA.

The labels are encoded as integer values from 0 to 4, corresponding to the five rice varieties in the order listed above.

3 Applied Methods and Algorithms

This section outlines the full pipeline applied to the rice image dataset, covering data preprocessing, dimensionality reduction using PCA, and supervised classification using multiple models. Each step is motivated by computational efficiency and model performance considerations.

3.1 Image Preprocessing

The original dataset images are resized to 32×32 RGB format, yielding an input tensor of shape $(N, 32, 32, 3)$ where N is the number of total images. Each image is then flattened into a vector of length 3072:

$$\mathbf{x}_i \in \mathbb{R}^{3072}, \quad i = 1, \ldots, N$$

The dataset is split into training and test sets using an 80-20 ratio:

$$\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$$

3.2 Standardization

Each input feature is standardized using **z-score normalization**:

$$z = \frac{x - \mu}{\sigma},$$

where μ and σ are the mean and standard deviation computed from the training set. Standardization ensures that each pixel intensity contributes equally to the PCA and classification algorithms.

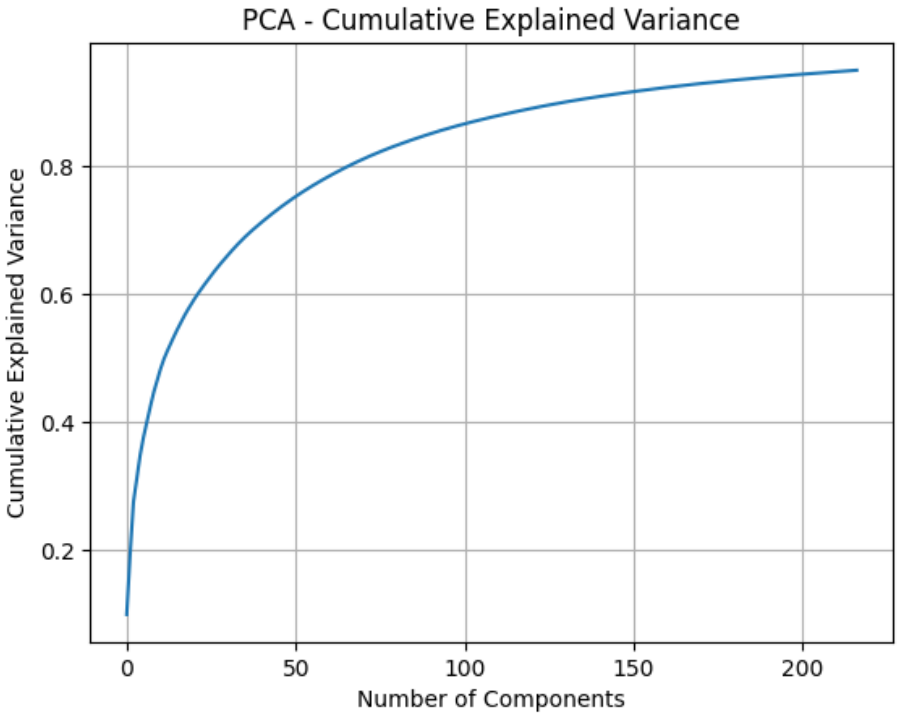
3.3 Principal Component Analysis (PCA)

To reduce the dimensionality of the dataset, PCA is applied to the standardized vectors. PCA finds an orthogonal transformation that maps the data to a lower-dimensional space while preserving as much variance as possible. The transformation matrix $\mathbf{W}$ is constructed from the top k eigenvectors of the covariance matrix:

$$\mathbf{Z} = \mathbf{X} \cdot \mathbf{W}, \quad \mathbf{W} \in \mathbb{R}^{d \times k}, \; d = 3072$$

The number of components k is chosen such that 95% of the variance is retained:

$$\sum_{i=1}^{k} \lambda_i \geq 0.95 \cdot \sum_{i=1}^{d} \lambda_i$$



This step substantially reduces computational complexity and helps mitigate overfitting. After PCA step each image has 217 features instead of 3072:

$$\mathbf{x}_i \in \mathbb{R}^{217}, \quad i = 1, \ldots, N \tag{1}$$

After resizing, scaling and applying PCA the images become blurry, as shown in Figure 3.

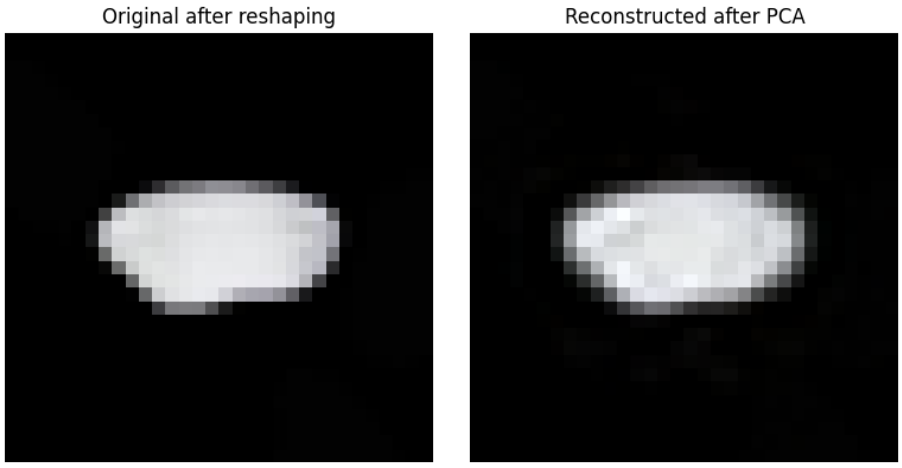


Figure 3: Comparison of rice image after resizing and after pca

3.4 Random Forest (RF)

RF is an ensemble method based on decision trees. Each tree is trained on a bootstrap sample with feature randomness. Final predictions are made via majority vote.

In our experiments, the following hyperparameters were used:

- `n_estimators` = 100 (number of trees)
- `max_features` = `"sqrt"` (square root of total features)
- `random_state` = 42 (for reproducibility).

A more a detailed explanation of the model can be found in Appendix A.

3.5 K-Nearest Neighbors (KNN)

KNN is a non-parametric method that classifies a data point based on the majority class of its k nearest neighbors in the feature space. The implementation was performed using the `KNeighborsClassifier` class from `scikit-learn` with the Euclidean distance and the fixed number of neighbors $k = 5$, which is a common default choice when no prior optimization is applied. Due to its lack of a training phase, KNN does not benefit from model persistence (i.e., saving to disk).

3.6 Logistic Regression (LR)

Logistic regression is a linear classifier that models the posterior probability using the softmax function:

$$P(y = c \mid \mathbf{x}) = \frac{e^{\mathbf{w}_c^\top \mathbf{x}}}{\sum_{j=1}^{C} e^{\mathbf{w}_j^\top \mathbf{x}}}$$

where C is the number of classes.

Each model is either trained or loaded from a pre-saved file, and then evaluated on the test set using accuracy and mean average precision (mAP). Logistic Regression was implemented via the `LogisticRegression` class from `scikit-learn`. It was fitted on the same training data as the other models. The implementation used L2 regularization and `lbfgs` solver, which is default in `scikit-learn`. The `max_iter` parameter was set to 1000 in order to ensure convergence.

3.7 Model Evaluation Metrics

- **Accuracy**: Proportion of correct predictions.
- **Confusion Matrix**: Illustrates class-wise performance.
- **Mean Average Precision (mAP)**: Calculated by taking mean of AP values over all classes

  $$mAP = \frac{1}{n} \sum_{k=1}^{k=C} AP_k,$$

  where C is the number of classes and AP_k is the average precision of class k.

4 Code Structure and Implementation

The implementation follows an object-oriented approach. Each stage of the data processing and analysis pipeline is encapsulated in a separate Python class to ensure clarity and reusability.

4.1 GoogleDataDownloader

This class is responsible for downloading and extracting the rice image dataset from Google Drive. If the dataset is not already available locally, the class:

- Downloads the ZIP archive using `gdown`
- Extracts it to a designated directory (`rice_data/`)
- Deletes the ZIP file to save space

It also includes visualization methods to plot sample images from each rice class, and to plot a pie-chart, showing balanced distribution of images between rice varieties.

4.2 ImageLoader

This class handles the loading and preprocessing of image data:

- Reads images from each subfolder, resizes them to 32×32 pixels, and stores them as NumPy arrays
- Assigns integer labels based on class names
- Caches the loaded data into `x.pkl` and `y.pkl` using `pickle`, or loads it from disk if already available

4.3 Preprocessor

This class prepares the image data for model training. It includes:

- Reshaping the image arrays into 2D arrays
- Splitting into training and testing sets
- Standardization using `StandardScaler`
- Dimensionality reduction with `PCA`, retaining 95% of the variance
- Saving/loading of the trained scaler and PCA objects using `pickle`

The class also provides utility functions for plotting explained variance and reconstructing images from PCA-transformed data.

4.4 ModelEvaluator

This class evaluates the performance of classification models. It is responsible for:

- Training or loading models
- Measuring accuracy and mean average precision (mAP)
- Displaying the confusion matrix

It ensures that models are saved and reused efficiently, and that the evaluation pipeline remains consistent across different classifiers.

5 Results

All classifiers showed good results due to high-quality pictures. Random Forest classifier scored accuracy = 0.9841 and mAP = 0.9717. As expected, confusion matrix in Figure 4 shows relatively frequent misclassification of Basmati and Jasmine, and Arborio and Karacadag.

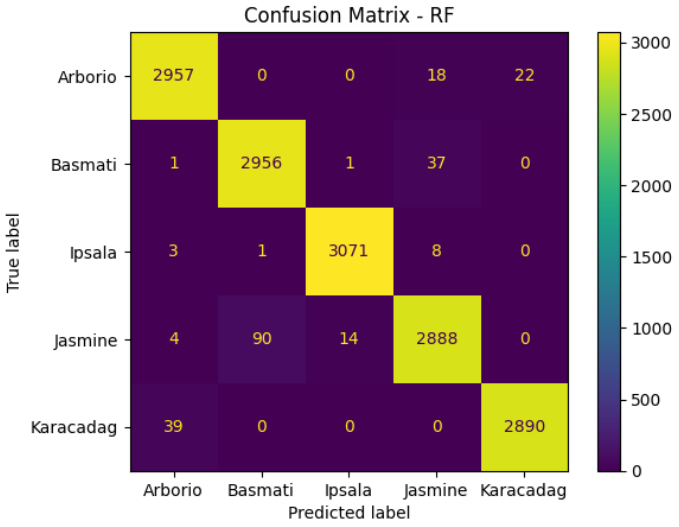


Figure 4: Confusion matrix of Random Forest classifier

Other models showed similar results.

K-nearest neighbors scored accuracy = 0.98 and mAP = 0.9644. The misclassification of Basmati and Jasmine became even more obvious.

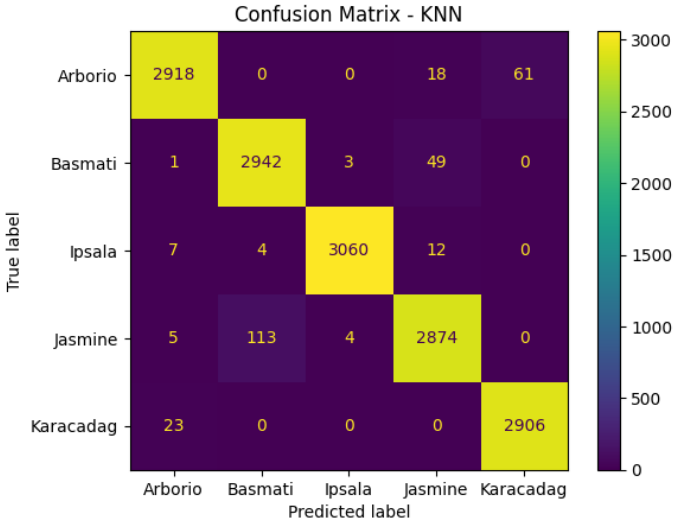


Figure 5: Confusion matrix of KNN

The last model used in the project – Logistic regression. It showed accuracy = 0.9847 and mAP = 0.9728.

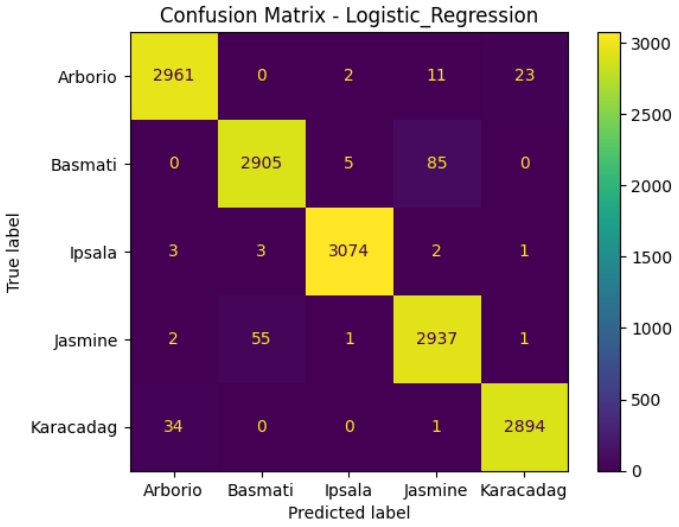


Figure 6: Confusion matrix of Logistic Regression

Therefore, logistic regression scored the highest mAP.

Although the results of all classical machine learning models are decent, deep learning models are more suitable for these tasks thanks to their scalability and

higher accuracy. Deep learning models do not require full dataset for training, which allows to save memory by implementing training in mini batches. In contrast, classical machine learning models are usually trained on full training dataset. During the training phase of the project memory overflow was one issue, which would not allow us to use a bigger image dataset.

A possible continuation of the project might be hyperparameter tuning or implementation of deep learning models.

6 Third-Party Python Libraries

Below is a list of third-party packages used in the project that are not listed among the allowed default libraries:

- **os**: Helps to navigate through folders on Google Drive
- **zipfile**: Used for extracting the dataset from a zip file
- **pickle**: Allows to upload/download datasets and models to/from Google Drive.
- **PIL (Pillow)**: Used for image processing tasks such as opening and resizing JPEG images.
- **scikit-learn (sklearn)**: Used for preprocessing (`StandardScaler`, `PCA`), model training and evaluation (`RandomForestClassifier`, `KNeighborsClassifier`, `LogisticRegression`, metrics and utilities).

Appendix A Random Forest Classifier

Random Forest is an ensemble learning method introduced by Breiman (2001), designed to improve the classification or regression accuracy of a single decision tree. It constructs a set of decision trees during training and outputs the class that is the mode of the predictions of the individual trees.

A.1 Decision Trees and Splitting Criteria

A single decision tree recursively partitions the input space using axis-aligned splits based on feature values. At each internal node, the best split is determined by minimizing an impurity measure. Two commonly used criteria are:

- **Gini Impurity:**

$$G(t) = 1 - \sum_{i=1}^{C} p_i^2$$

  where p_i is the proportion of class i instances at node t, and C is the number of classes.

- **Entropy (Information Gain):**

$$H(t) = -\sum_{i=1}^{C} p_i \log_2 p_i$$

In the project Gini Impurity was used, which is the default option for `RandomForestClassifier`. The optimal feature and threshold at each node are chosen to maximize the reduction in impurity (e.g., maximize information gain or decrease in Gini impurity).

A.2 Bagging (Bootstrap Aggregating)

Random Forest builds each tree on a different bootstrap sample $D_b \subseteq D$, where D is the full training dataset. Each D_b is generated by sampling n times from D with replacement.

Let $h_1(x), \ldots, h_T(x)$ be the predictions of the T trees. The final prediction $H(x)$ for classification is obtained by majority voting:

$$H(x) = \arg\max_y \sum_{t=1}^{T} \mathbf{1}_{[h_t(x)=y]}$$

This technique reduces variance without increasing bias, resulting in a more robust model compared to individual trees.

A.3 Random Feature Selection

To further decorrelate trees and reduce overfitting, Random Forest introduces additional randomness by selecting only a random subset of features $m \ll p$ (where p is the total number of features) when determining the best split at each node.

Typically, $m = \sqrt{p}$ for classification tasks.

A.4 Bias-Variance Perspective

The expected generalization error of an ensemble can be expressed as:

$$\mathbb{E}[(f(x) - \hat{f}(x))^2] = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

Random Forest significantly reduces the variance term (the sensitivity to training data) by averaging many uncorrelated models, while keeping the bias low.

A.5 Summary of the Algorithm

Given training data $D = \{(x_i, y_i)\}_{i=1}^n$, Random Forest works as follows:

1. For $t = 1, \ldots, T$:
   (a) Draw a bootstrap sample D_t from D
   (b) Train a decision tree h_t on D_t, choosing the best split at each node from a random subset of features
2. Output the aggregated prediction $H(x)$ via majority vote:

$$H(x) = \arg\max_y \sum_{t=1}^{T} \mathbf{1}_{[h_t(x)=y]}$$