Fruits Recognition

Image processing

Nouran Hassan Ahmed *2022/00062*

TABLE OF CONTENTS

DATASET

- Here is the link of the dataset that I have chosen:

  https://www.kaggle.com/datasets/utkarshsaxenadn/fruits-classification

Dataset Properties

This dataset consists of images of various fruits and vegetables, offering a rich collection for image recognition tasks.

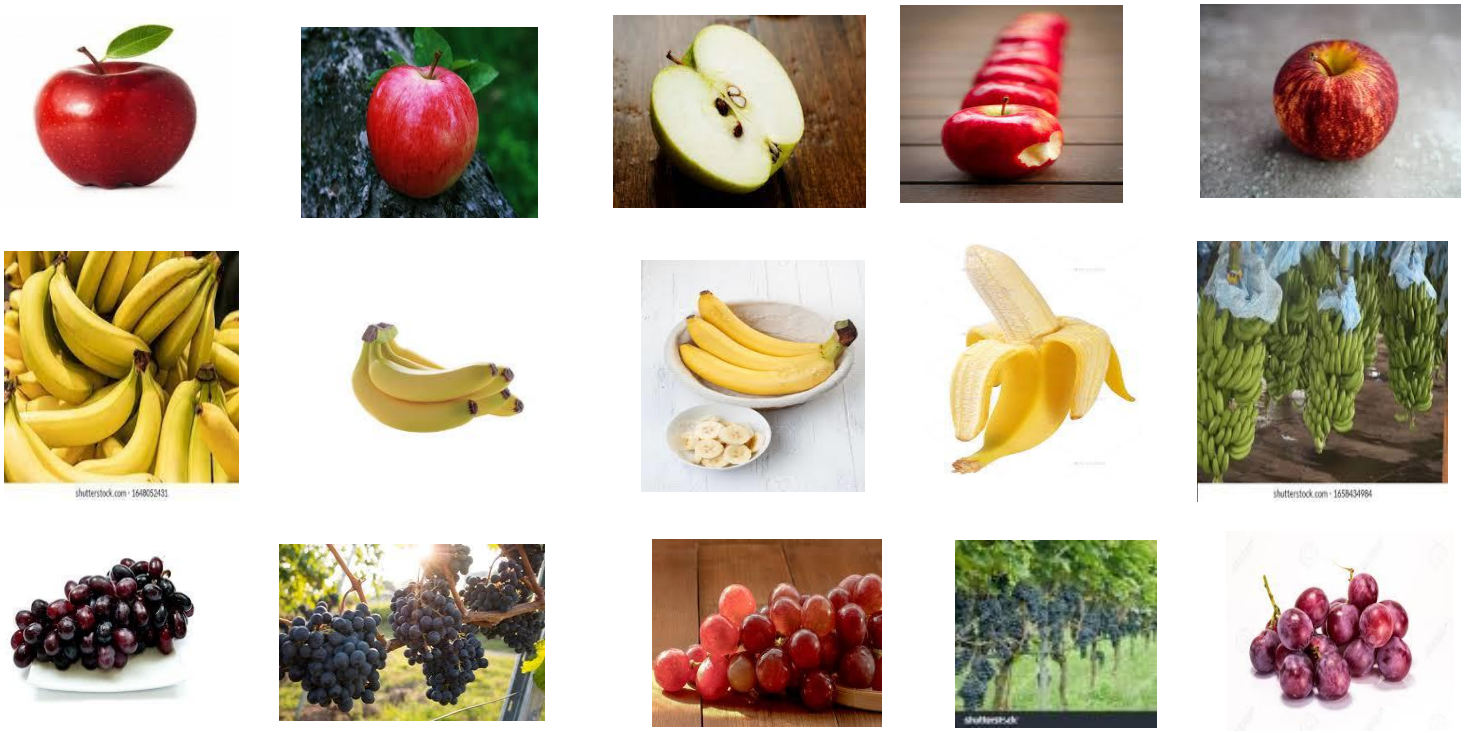
Categories

1. **Fruits :** Banana, Apple, Grapes, Strawberry, Mango

Dataset Organization

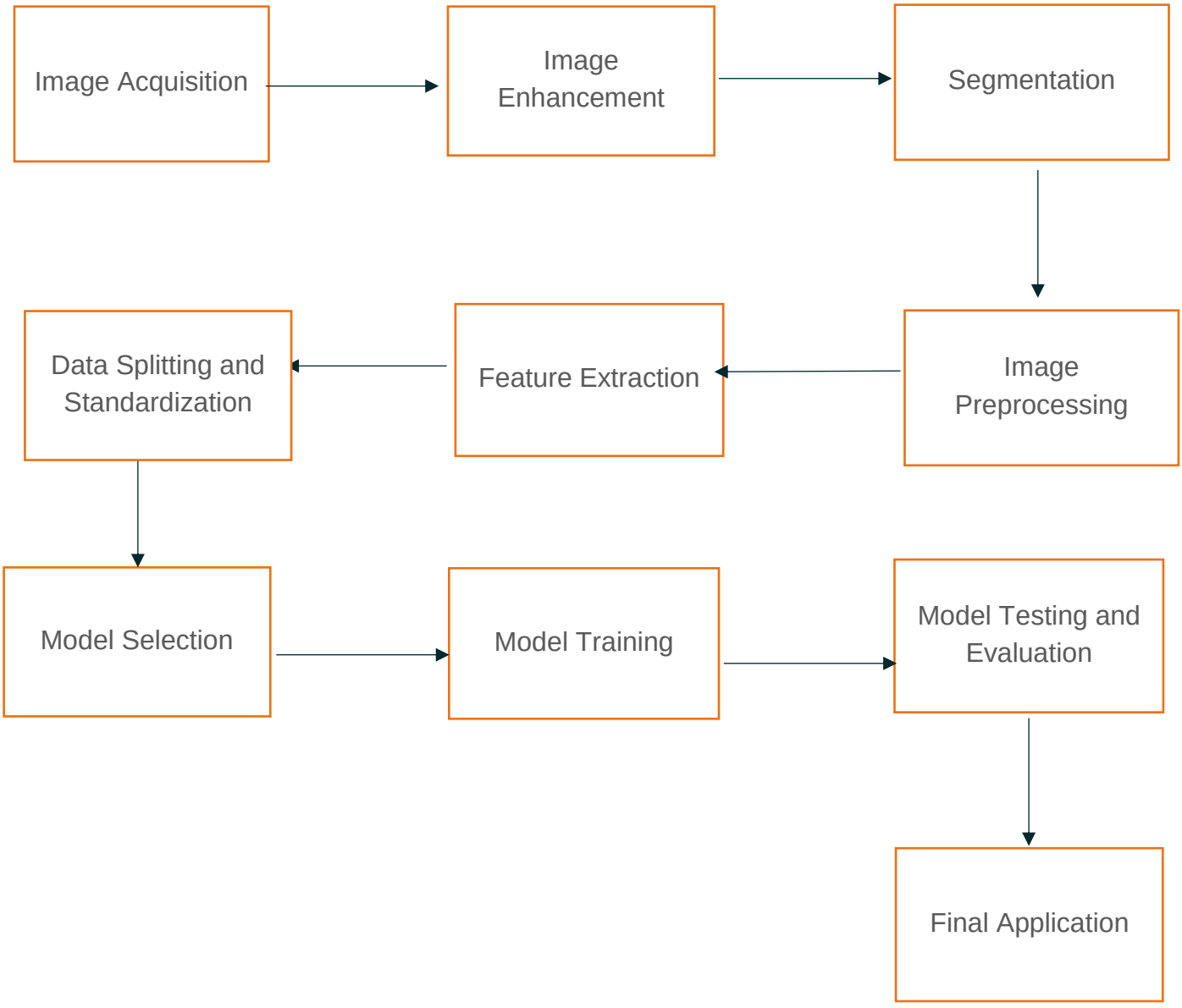
1. Each class contains **2000 images**, resulting in a total of **10,000 images in the dataset.**
2. The data is split into **three sets: training, validation, and testing**.

- Sample of dataset images:



DIAGRAM

- This diagram shows the sequence of steps I plan to follow in the project



DESCRIPTION OF EACH STEP

1. Image Acquisition

We already have the dataset ready, and its properties are detailed on page 3.
In this step, we acquire the image data from the specified path using the OpenCV `cv2.imread()` function. This function reads the image from the file and loads it into memory, making it available for further processing and analysis.

---

2. Image Enhancement

To improve the robustness of the model, we apply the following transformations:

- **Rotation**: The image is randomly rotated by an angle between -30° and 30°. This simulates varying orientations of the object.
- **Flipping**: The image is randomly flipped horizontally with a 50% chance. This ensures the model generalizes well for flipped objects.
- **Scaling**: The image is randomly scaled by a factor between 80% and 120%. This simulates different distances between the object and the camera.

---

3. Segmentation

Color-Based Segmentation:
We convert the image to a different color space, such as HSV, which is often better for segmenting colored objects.

- **Thresholding**: Apply a color threshold to isolate the fruit. For instance, you can isolate red fruits using a specific range of hues.
- **Edge Detection**: Use Canny edge detection to find the boundaries of the fruit.
- **Masking**: Create a binary mask where the fruit areas are set to 1 and the rest to 0. This helps isolate the fruit from the background.

---

4. Image Preprocessing

In this step, we prepare the dataset for machine learning by performing the following actions:

- **Resizing**: Each image is resized to a target size of (64, 64) to ensure consistent input dimensions for the model.
- **Normalization**: The pixel values of the image are normalized by dividing by 255.0, which scales them to a range of [0, 1]. This helps the model learn better by ensuring uniform data scale.
- **Flattening**: The image is flattened into a 1D vector so it can be used as a feature vector for machine learning models. Each pixel's value is transformed into a single row vector.

---

5. Feature Extraction

In this step, we manually extract features that capture the essence of the fruit images:

- **Color Histogram**: We calculate the frequency distribution of pixel values for each color channel (Red, Green, Blue). This helps capture the color patterns in the image, which are crucial for identifying fruits.
- **Normalization**: The histogram is normalized (so that the sum of the histogram values equals 1) and flattened into a 1D vector for use in machine learning models.

Additionally, we associate each image with its corresponding label. The labels are extracted from the folder names (which represent the fruit categories). For example, if an image is in a folder called "apple", the label for that image is "apple".

---

6. Data Splitting and Standardization

- **Data Splitting**: The dataset is split into training and validation sets. 80% of the data is used for training, and 20% is reserved for validation. This ensures that the model is trained on one set of data and evaluated on a separate set, preventing overfitting.
- **Standardization**: The feature data is standardized using `StandardScaler` to ensure the features have a mean of 0 and a standard deviation of 1. This helps models that rely on distance metrics (such as Random Forest) perform better.

---

7. Model Selection

We choose the Random Forest Classifier for this task due to its strengths with the given data:

- **Random Forest**: This algorithm is well-suited for tabular data with extracted features. It works well with multiple classes, which is ideal for the Fruits-360 dataset that contains several fruit categories.
- **Advantages**: Random Forest is robust and provides good performance even without extensive parameter tuning.

---

8. Model Training

In this step, we train the selected machine learning model using the prepared feature set and labeled data:

- A Random Forest Classifier is initialized with 100 trees (`n_estimators=100`) and trained on the scaled training data (`X_train_scaled`) along with their corresponding labels (`y_train`).
- Random Forest is an ensemble learning method that combines multiple decision trees to improve prediction accuracy and robustness.

---

9. Model Testing and Evaluation

After training the model, we evaluate its performance using a separate test set:

- **Metrics**: We use the accuracy score and classification report (precision, recall, F1-score) to assess the model's performance. These metrics help determine how well the model is predicting the correct fruit types.
- **Confusion Matrix**: A confusion matrix is generated to visualize the performance of the model. It shows how many instances were correctly or incorrectly classified for each fruit class. The heatmap makes it easier to identify where the model is making errors.
- **Grid Search**: We use grid search to find the best hyperparameters for the Random Forest model. This involves testing different combinations of hyperparameters (e.g., the number of trees `n_estimators`, maximum depth `max_depth`, etc.) using cross-validation to find the best-performing model.

---

10. Final Application

Once we find the best model through grid search, we save the model to a file using `joblib.dump()`. This allows us to reuse the model later without the need for retraining.