FRUITS RECOGNITION

Nouran Hassan Ahmed 2022/00062

Image processing project

Contents

Introduction

Accurately identifying fruits from images is a common challenge in computer vision with applications ranging from automated grocery checkouts to dietary tracking systems. This project aims to develop a fruit recognition system using image processing techniques and machine learning algorithms. The primary objectives are to preprocess the data effectively, extract meaningful features, train robust models, and create a user-friendly application capable of real-time fruit classification.

Objective

This document outlines the implementation of a fruit recognition system using image processing techniques. The system utilizes the Fruits dataset, which contains high-quality images of various fruits, categorized into training and testing sets. The goal is to create a robust pipeline for feature extraction, model training, and evaluation, culminating in a functional application.

Methodologies

Dataset

The dataset used for this project is available on Kaggle and can be accessed through the following link:
https://www.kaggle.com/datasets/moltean/fruits

Dataset Overview

The dataset comprises a collection of images featuring a variety of fruits, divided into three separate directories: training, validation, and testing. Each directory contains labeled images organized by class.

This dataset was selected for its diversity and well-organized structure, making it ideal for developing and evaluating the fruit recognition model.

The Diagram

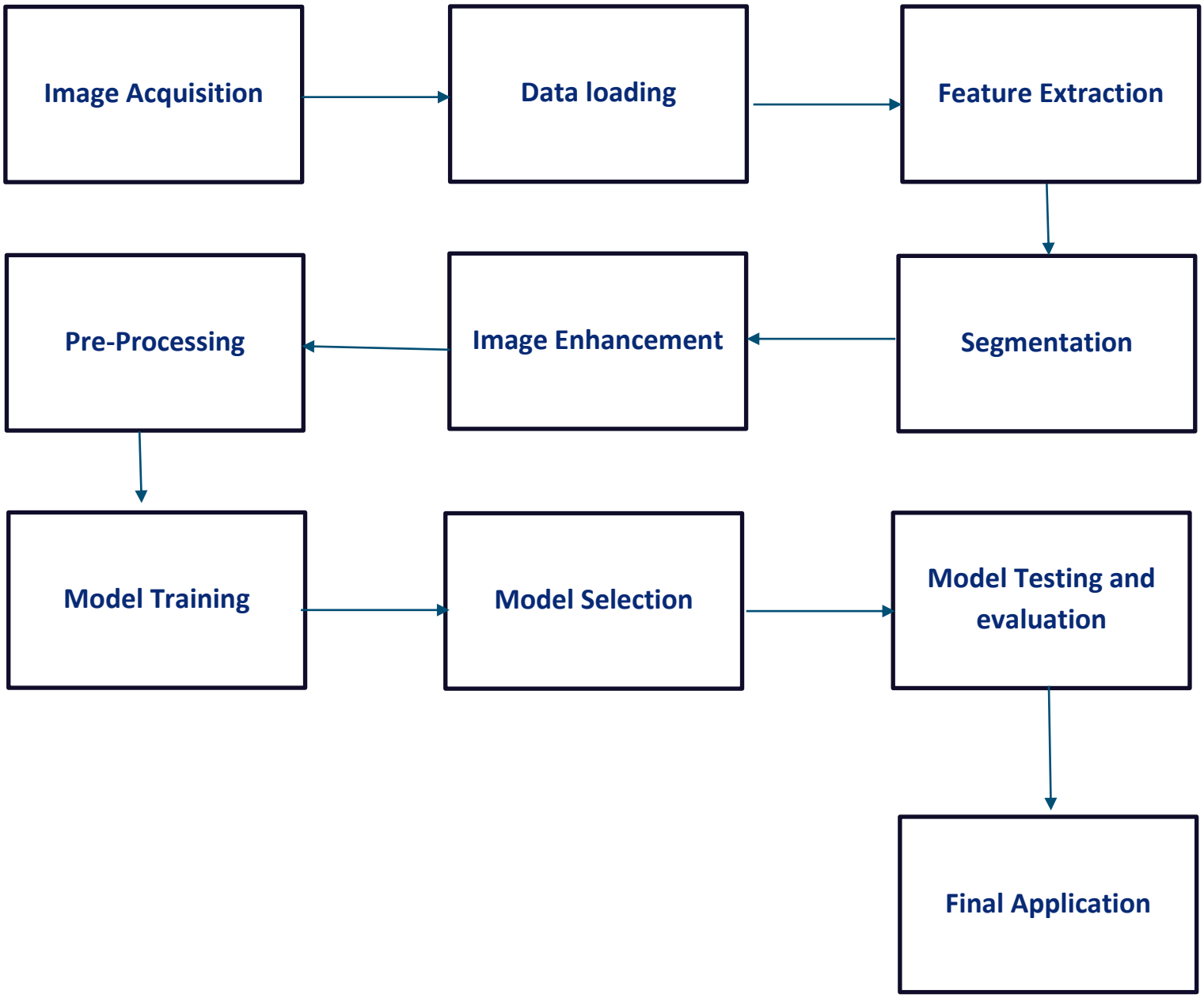


Image Acquisition
Data loading
Feature Extraction
Segmentation
Image Enhancement
Pre-Processing
Model Training
Model Selection
Model Testing and evaluation
Final Application

Description of Each Step:

1. Image Acquisition

The dataset is divided into training and testing directories. Each directory contains subfolders for each fruit class. The code verifies the existence of these directories.

```
train path = "/root/.cache/kagglehub/datasets/.../Training"
test path = "/root/.cache/kagglehub/datasets/.../Test"
```

2. Loading Data

Functions were implemented to load the images and their corresponding labels from the dataset. The images are stored in NumPy arrays for further processing for the training and testing files

Code Snippet

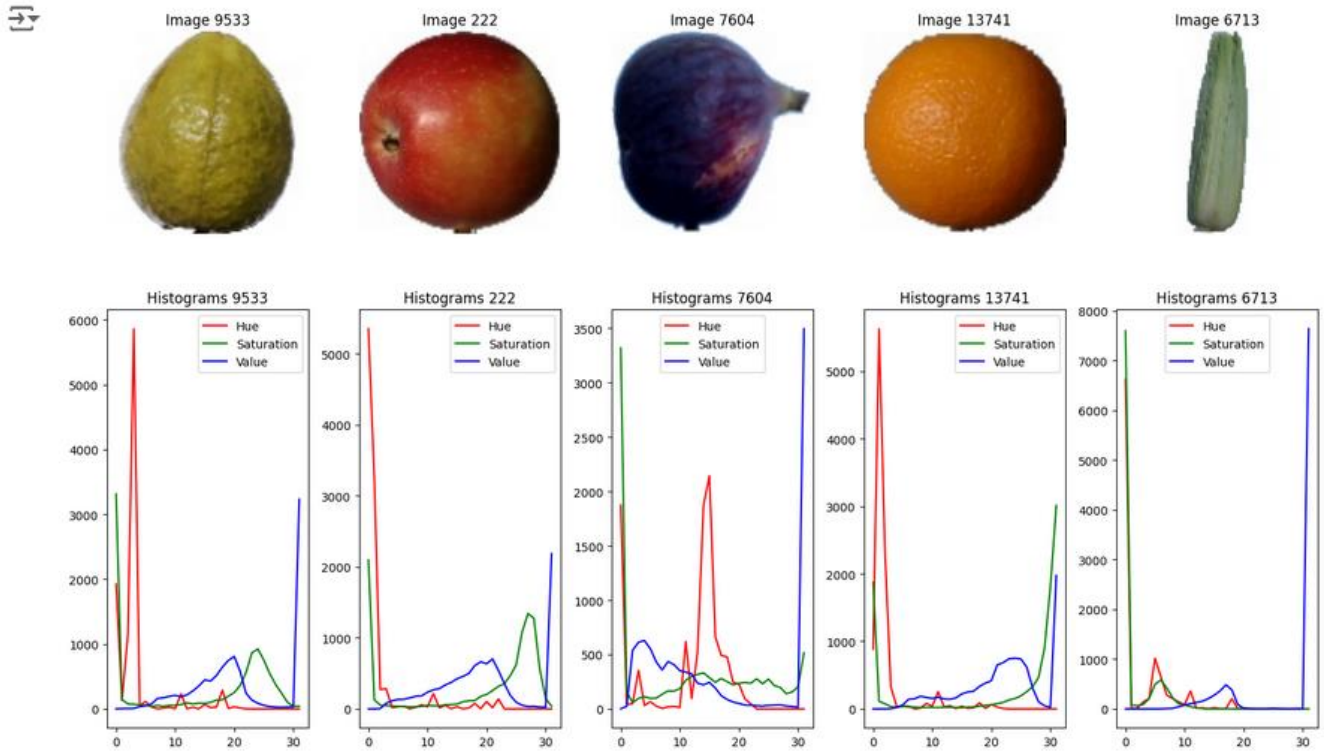
```python
def load_train_data(train_path):
    features, labels = [], []
    classes = sorted(os.listdir(train_path))
    for class_name in classes:
        class_path = os.path.join(train_path, class_name)
        if os.path.isdir(class_path):
            for img_name in os.listdir(class_path):
                img_path = os.path.join(class_path, img_name)
                image = cv2.imread(img_path)
                if image is not None:
                    features.append(image)
                    labels.append(class_name)
    return np.array(features), np.array(labels)
```

3. Feature Extraction

- **Method**: Color histograms.
- Images resized to 100x100 pixels and converted to HSV color space.
- Histograms computed for Hue, Saturation, and Value channels.
- Normalized and concatenated for feature representation.

Code Snippet

```python
def extract_color_histograms(images, bins=32):
    all_histograms = []
    for img in tqdm(images, desc="Extracting Color Histograms", unit="image"):
        resized_img = cv2.resize(img, (100, 100))  # Resize to a fixed size
        hsv_img = cv2.cvtColor(resized_img, cv2.COLOR_BGR2HSV)  # Convert to HSV color space
        hist_hue = cv2.calcHist([hsv_img], [0], None, [bins], [0, 256]).flatten()
        hist_saturation = cv2.calcHist([hsv_img], [1], None, [bins], [0, 256]).flatten()
        hist_value = cv2.calcHist([hsv_img], [2], None, [bins], [0, 256]).flatten()
        hist = np.concatenate([hist_hue, hist_saturation, hist_value])
        hist = hist / hist.sum()  # Normalize to get a probability distribution
        all_histograms.append(hist)
    return np.array(all_histograms)
```



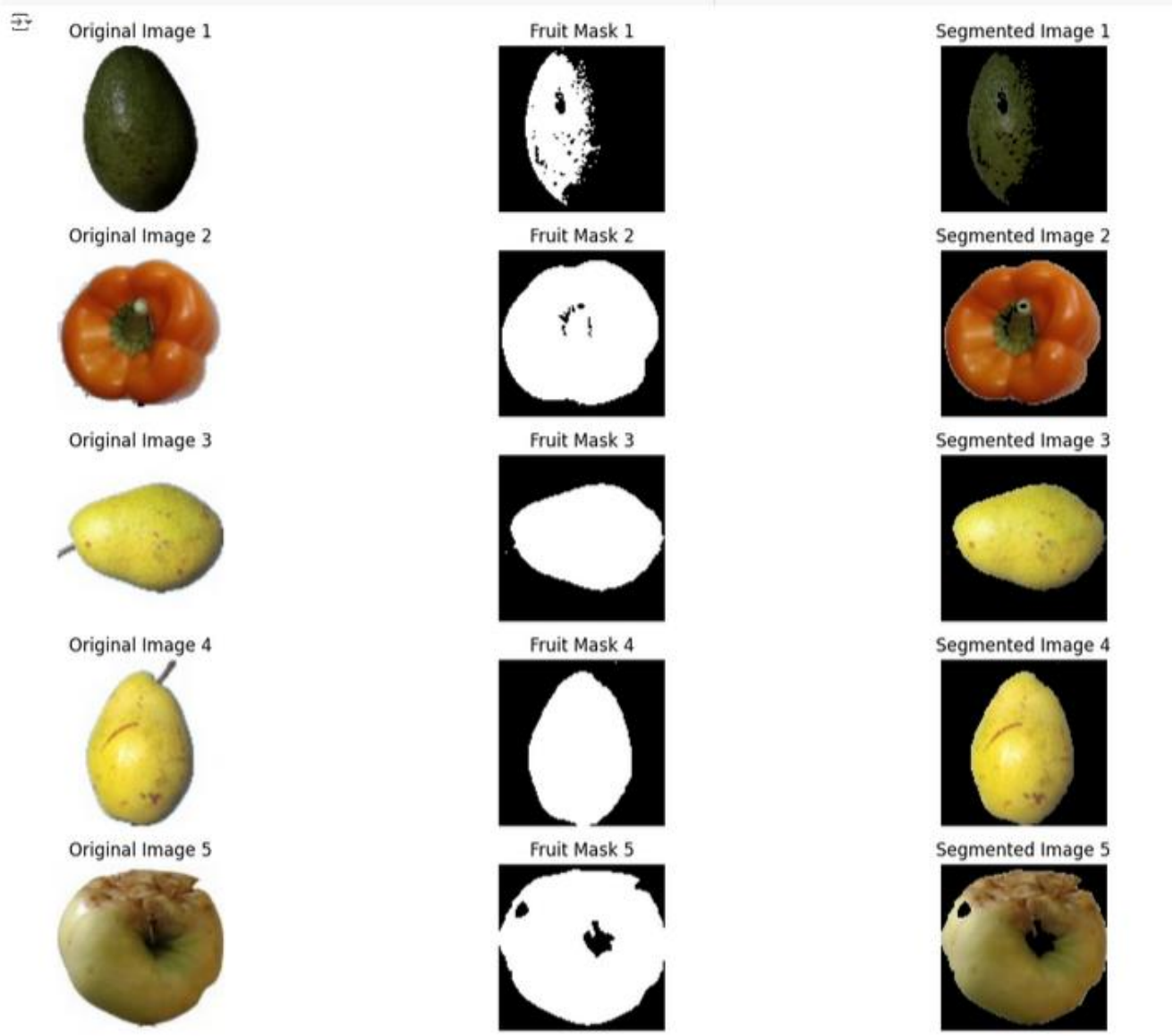
5. Image Segmentation

Image segmentation was performed using color thresholds to isolate the fruit from the background. The segmentation process excluded backgrounds and isolated the colored regions corresponding to the fruit. As we converted the colored images to HSV for the color improvement then we started creating the upper bound and lower bound

Code Snippet

```
# Function for image segmentation using color thresholds (to keep the fruit colored)
def segment_image_with_color(img, hue_min=0, hue_max=179, sat_min=50, sat_max=255, val_min=50, val_max=255):
    """
    Segment an image background and fruit in the foreground using color thresholds.
    Excludes pixels (high value and low saturation) and isolates the fruit while keeping colors.
    """
    # Convert the image to HSV color space
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Create a mask to exclude background (where value is high and saturation is low)
    lower_bound = np.array([0, 0, 200])  # High value to exclude background
    upper_bound = np.array([179, 50, 255])  # Low saturation and high value for  background

    # Mask for non-white pixels (background)
    mask = cv2.inRange(hsv_img, lower_bound, upper_bound)
    mask= cv2.bitwise_not(mask)

    # Define the thresholds for the fruit colors (tune these values for different fruits)
    lower_bound_fruit = np.array([hue_min, sat_min, val_min])
    upper_bound_fruit = np.array([hue_max, sat_max, val_max])

    # Create the fruit mask using the thresholds
    mask_fruit = cv2.inRange(hsv_img, lower_bound_fruit, upper_bound_fruit)

    # Combine the masks: Focus on fruit
    final_mask = cv2.bitwise_and(mask_fruit, mask)

    # Apply the final mask to the original image to isolate the fruit (colored)
    colored_segmented_img = cv2.bitwise_and(img, img, mask=final_mask)

    return final_mask, colored_segmented_img
```

Visualization:


Original Image 1
Fruit Mask 1
Segmented Image 1
Original Image 2
Fruit Mask 2
Segmented Image 2
Original Image 3
Fruit Mask 3
Segmented Image 3
Original Image 4
Fruit Mask 4
Segmented Image 4
Original Image 5
Fruit Mask 5
Segmented Image 5

6. Image Enhancement

- Enhances brightness, contrast, and sharpness.
- Techniques include histogram equalization and noise reduction, denoising, and sharpening.

Code Snippet

```python
def enhance_image(img):
    """
    Apply image enhancement and restoration techniques to improve the segmented image.
    Techniques:
    1. Brightness and Contrast Adjustment
    2. Histogram Equalization
    3. Denoising
    4. Sharpening
    """
    # 1. Brightness and Contrast Adjustment
    alpha = 1.2  # Contrast control (1.0-3.0)
    beta = 30    # Brightness control (0-100)
    enhanced_img = cv2.convertScaleAbs(img, alpha=alpha, beta=beta)

    # 2. Histogram Equalization (only for luminance channel in YUV)
    yuv_img = cv2.cvtColor(enhanced_img, cv2.COLOR_BGR2YUV)
    yuv_img[:, :, 0] = cv2.equalizeHist(yuv_img[:, :, 0])
    enhanced_img = cv2.cvtColor(yuv_img, cv2.COLOR_YUV2BGR)

    # 3. Noise Removal (Denoising)
    denoised_img = cv2.fastNlMeansDenoisingColored(enhanced_img, None, 10, 10, 7, 21)

    # 4. Sharpening
    kernel = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, -1, 0]])
    sharpened_img = cv2.filter2D(denoised_img, -1, kernel)

    return sharpened_img
```

Visualization of the Segmentation and the Enhancement together:

6. Image Preprocessing

Feature Standardization

Extracted features are standardized using `Standard Scaler` to ensure a uniform range, optimizing them for model training and then update the pipeline with the preprocessing to the training and testing data

Code Snippet:

```
# Preprocessing: Standardizing features
def preprocess_features(train_features, test_features):
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(train_features)
    X_test_scaled = scaler.transform(test_features)
    return X_train_scaled, X_test_scaled
```

```
# Updated Pipeline with Preprocessing
print("Applying segmentation and enhancement to the training data...")
segmented_enhanced_train = [enhance_image(segment_image_with_color(img, 0, 179, 50, 255, 50, 255)[1]) for img in X_train]
```

```
Applying segmentation and enhancement to the training data...
```

```
print("Applying segmentation and enhancement to the testing data...")
segmented_enhanced_test = [enhance_image(segment_image_with_color(img, 0, 179, 50, 255, 50, 255)[1]) for img in X_test]
```

```
Applying segmentation and enhancement to the testing data...
```

```
print("Extracting color histogram features from training data...")
hist_train = extract_color_histograms(segmented_enhanced_train)
```

```
Extracting color histogram features from training data...
Extracting Color Histograms: 100%|██████████| 70491/70491 [00:13<00:00, 5178.84image/s]
```

```
print("Extracting color histogram features from testing data...")
hist_test = extract_color_histograms(segmented_enhanced_test)
```

```
Extracting color histogram features from testing data...
Extracting Color Histograms: 100%|██████████| 23619/23619 [00:03<00:00, 5933.02image/s]
```

```
print("Scaling features...")
X_train_scaled, X_test_scaled = preprocess_features(hist_train, hist_test)
```

```
Scaling features...
```

Testing and Results

7. Model Training, selection and Evaluation

Accuracy of the Models before Performing hyperparameter tuning using cross-validation and Random Search

- **SVM:**

```
[ ]  # Training and evaluating SVM model
     print("Training SVM model...")
     svm_classifier = SVC(kernel='linear', random_state=42)
     svm_classifier.fit(X_train_scaled, y_train)

     print("Evaluating SVM model...")
     y_pred_svm = svm_classifier.predict(X_test_scaled)
     svm_accuracy = accuracy_score(y_test, y_pred_svm)
     print("SVM Accuracy:", svm_accuracy)
     print("SVM Classification Report:")
     print(classification_report(y_test, y_pred_svm))
```

```
Training SVM model...
Evaluating SVM model...
SVM Accuracy: 0.9413607688725179
```

- **KNN**

```
# Training and evaluating KNN model with Euclidean distance
print("Training KNN model with Euclidean distance...")
knn_classifier = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn_classifier.fit(X_train_scaled, y_train)

print("Evaluating KNN model...")
y_pred_knn = knn_classifier.predict(X_test_scaled)
knn_accuracy = accuracy_score(y_test, y_pred_knn)
print("KNN Accuracy:", knn_accuracy)
print("KNN Classification Report:")
print(classification_report(y_test, y_pred_knn))
```

```
Training KNN model with Euclidean distance...
Evaluating KNN model...
KNN Accuracy: 0.9178203988314493
```

- **Random Forest**

```
# Training and evaluating Random Forest model
print("Training Random Forest model...")
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train_scaled, y_train)

print("Evaluating Random Forest model...")
y_pred_rf = rf_classifier.predict(X_test_scaled)
rf_accuracy = accuracy_score(y_test, y_pred_rf)
print("Random Forest Accuracy:", rf_accuracy)
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

```
Training Random Forest model...
Evaluating Random Forest model...
Random Forest Accuracy: 0.9507176425758923
```

Classification Report -> a part of one of the classification reports you can find the rest in the code implementation

```
Random Forest Classification Report:
                     precision    recall  f1-score   support

            Apple 6       0.79      1.00      0.88       157
    Apple Braeburn 1       0.75      0.79      0.77       164
Apple Crimson Snow 1       0.92      0.97      0.95       148
     Apple Golden 1        0.88      0.99      0.93       160
     Apple Golden 2        1.00      1.00      1.00       164
     Apple Golden 3        1.00      1.00      1.00       161
Apple Granny Smith 1       1.00      1.00      1.00       164
   Apple Pink Lady 1       0.78      0.82      0.80       152
         Apple Red 1       0.73      0.80      0.76       164
         Apple Red 2       0.86      0.95      0.90       164
         Apple Red 3       0.83      0.90      0.87       144
Apple Red Delicious 1      0.92      1.00      0.96       166
  Apple Red Yellow 1       0.96      0.99      0.98       164
  Apple Red Yellow 2       0.87      1.00      0.93       219
         Apple hit 1       0.93      1.00      0.96       234
           Apricot 1       1.00      0.99      0.99       164
           Avocado 1       1.00      0.80      0.89       143
      Avocado ripe 1       0.90      0.80      0.85       166
            Banana 1       0.97      0.99      0.98       166
Banana Lady Finger 1       0.97      1.00      0.98       152
       Banana Red 1        1.00      0.80      0.89       166
         Beetroot 1        0.69      0.93      0.79       150
        Blueberry 1        1.00      1.00      1.00       154
    Cabbage white 1        1.00      1.00      1.00        47
      Cactus fruit 1       0.98      1.00      0.99       166
```

Models Accuracy after Performing hyperparameter tuning using cross-validation and Random Search implementation

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best Random Forest model parameters: {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_depth': None, 'bootstrap': False}
Random Forest Accuracy: 0.9530039375079385
Best SVM model parameters: {'kernel': 'linear', 'gamma': 'auto', 'C': 100}
SVM Accuracy: 0.9459333587366103
Best KNN model parameters: {'weights': 'uniform', 'n_neighbors': 5, 'algorithm': 'ball_tree'}
KNN Accuracy: 0.9178203988314493
The best performing model is Random Forest with accuracy: 0.9530
```

- The highest accuracy reached was 95.3% using random search with the model Random Forest

Implementation:

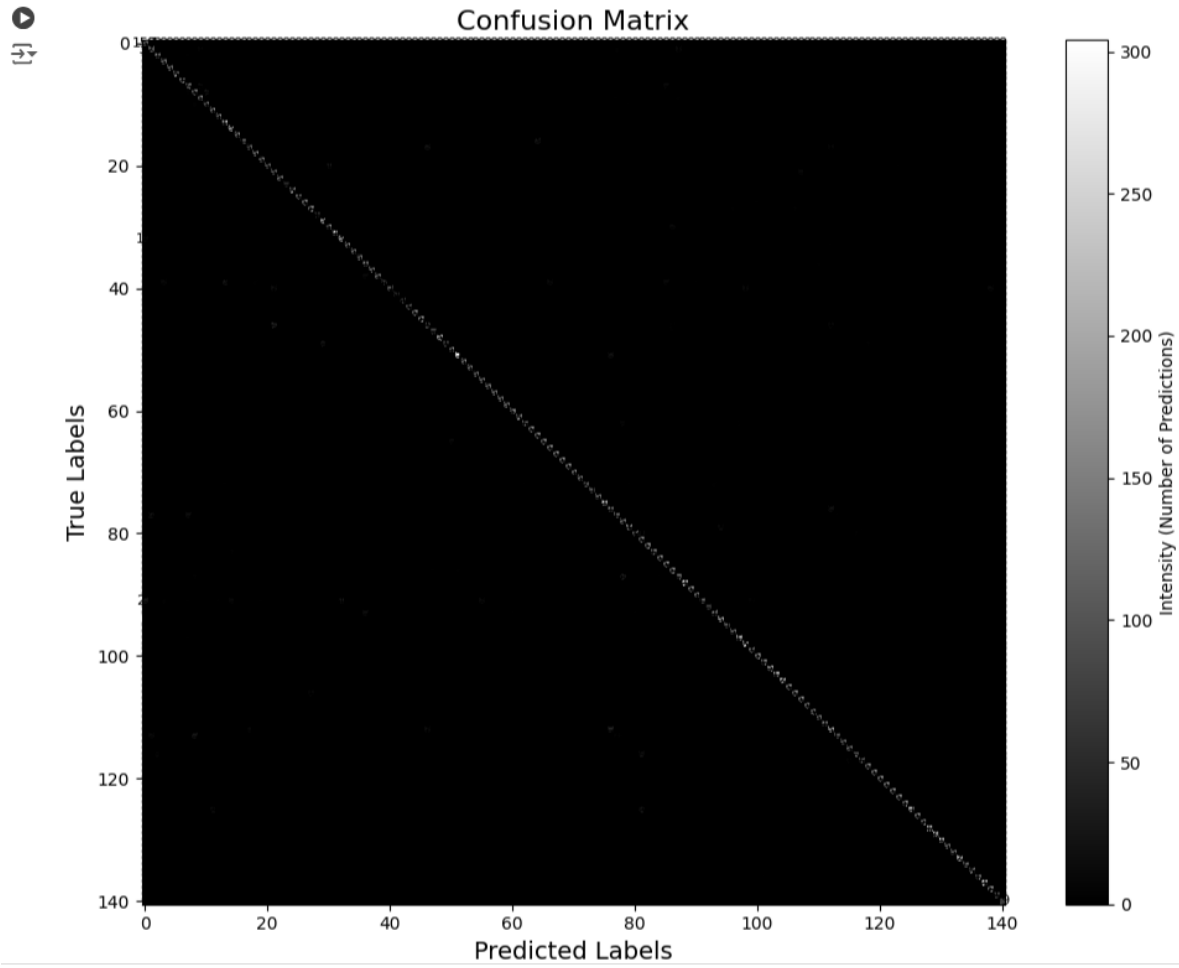
Various machine learning models were trained on the extracted features, including Support Vector Machines (SVMs), Random Forests, and k-Nearest Neighbors (k-NN). The training process involved hyperparameter tuning using cross-validation. Models were evaluated based on accuracy, precision, recall, and F1-score. The SVM model with a linear kernel achieved the highest performance metrics. Confusion matrices and classification reports were generated to assess performance.

```
# Hyperparameter search space
param_dist_rf = {
    'n_estimators': [50, 100, 200, 500],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

param_dist_svm = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto'],
}

param_dist_knn = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

# Define classifiers
svm_classifier = SVC(random_state=42)
knn_classifier = KNeighborsClassifier()
rf_classifier = RandomForestClassifier(random_state=42)

# Perform Random Search on Random Forest
random_search_rf = RandomizedSearchCV(rf_classifier, param_distributions=param_dist_rf, n_iter=10,
                                      cv=3, random_state=42, n_jobs=-1, verbose=1)
random_search_rf.fit(X_train_scaled, y_train)

# Perform Random Search on SVM
random_search_svm = RandomizedSearchCV(svm_classifier, param_distributions=param_dist_svm, n_iter=10,
                                       cv=3, random_state=42, n_jobs=-1, verbose=1)
random_search_svm.fit(X_train_scaled, y_train)

# Perform Random Search on KNN
random_search_knn = RandomizedSearchCV(knn_classifier, param_distributions=param_dist_knn, n_iter=10,
                                       cv=3, random_state=42, n_jobs=-1, verbose=1)
random_search_knn.fit(X_train_scaled, y_train)

# Evaluate the best models
print("Best Random Forest model parameters:", random_search_rf.best_params_)
best_rf_model = random_search_rf.best_estimator_
y_pred_rf = best_rf_model.predict(X_test_scaled)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))

print("Best SVM model parameters:", random_search_svm.best_params_)
best_svm_model = random_search_svm.best_estimator_
y_pred_svm = best_svm_model.predict(X_test_scaled)
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))

print("Best KNN model parameters:", random_search_knn.best_params_)
best_knn_model = random_search_knn.best_estimator_
y_pred_knn = best_knn_model.predict(X_test_scaled)
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))

# Compare the models based on their accuracy
accuracies = {
    'Random Forest': accuracy_score(y_test, y_pred_rf),
    'SVM': accuracy_score(y_test, y_pred_svm),
    'KNN': accuracy_score(y_test, y_pred_knn)
}

best_model_name = max(accuracies, key=accuracies.get)
print(f"The best performing model is {best_model_name} with accuracy: {accuracies[best_model_name]:.4f}")
```

Confusion Matrix

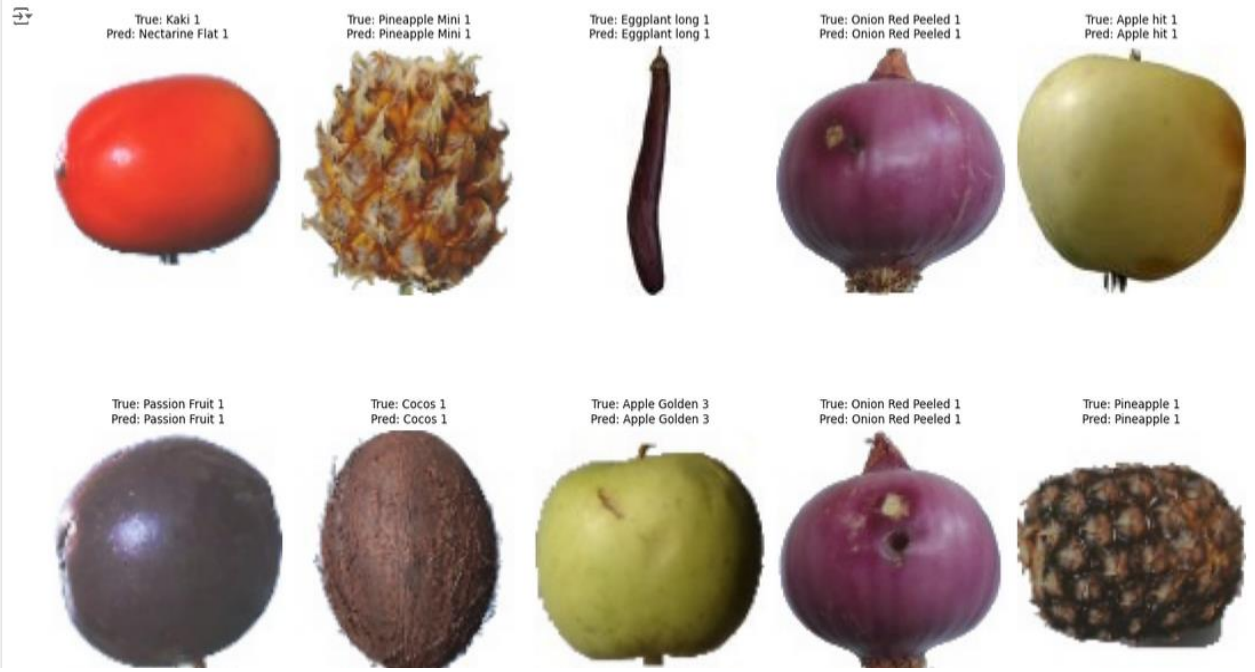
```
[ ]  # Confusion matrix
     def plot_confusion_matrix(y_true, y_pred, class_names):
         cm = confusion_matrix(y_true, y_pred, labels=class_names)
         plt.figure(figsize=(10, 8))
         plt.imshow(cm, cmap='gray', interpolation='nearest')
         plt.title('Confusion Matrix', fontsize=16)
         plt.xlabel('Predicted Labels', fontsize=14)
         plt.ylabel('True Labels', fontsize=14)
         for i in range(len(class_names)):
             for j in range(len(class_names)):
                 plt.text(j, i, cm[i, j], ha='center', va='center', color='black', fontsize=8)
         plt.tight_layout()
         plt.colorbar(label="Intensity (Number of Predictions)")
         plt.show()
```

```
[ ]  # Generate confusion matrix for the best model
     class_names = np.unique(y_test)
     plot_confusion_matrix(y_test, y_pred_best, class_names)
```



Testing the prediction using the testing images

```python
# Visualize random predictions
def visualize_predictions(X_test, y_test, y_pred, num_samples=10):
    plt.figure(figsize=(15, 10))
    indices = random.sample(range(len(X_test)), num_samples)
    for i, idx in enumerate(indices):
        plt.subplot(2, 5, i + 1)
        plt.imshow(cv2.cvtColor(X_test[idx], cv2.COLOR_BGR2RGB))
        true_label = y_test[idx]
        predicted_label = y_pred[idx]
        plt.title(f"True: {true_label}\nPred: {predicted_label}", fontsize=10)
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Visualize random predictions for the best model
visualize_predictions(X_test, y_test, y_pred_best, num_samples=10)
```



Saving the best model to use it in the website

```python
import pickle

# Save the best Random Forest model
with open('best_rf_model.pkl', 'wb') as file:
    pickle.dump(best_rf_model, file)

print("Best Random Forest model saved successfully!")
```

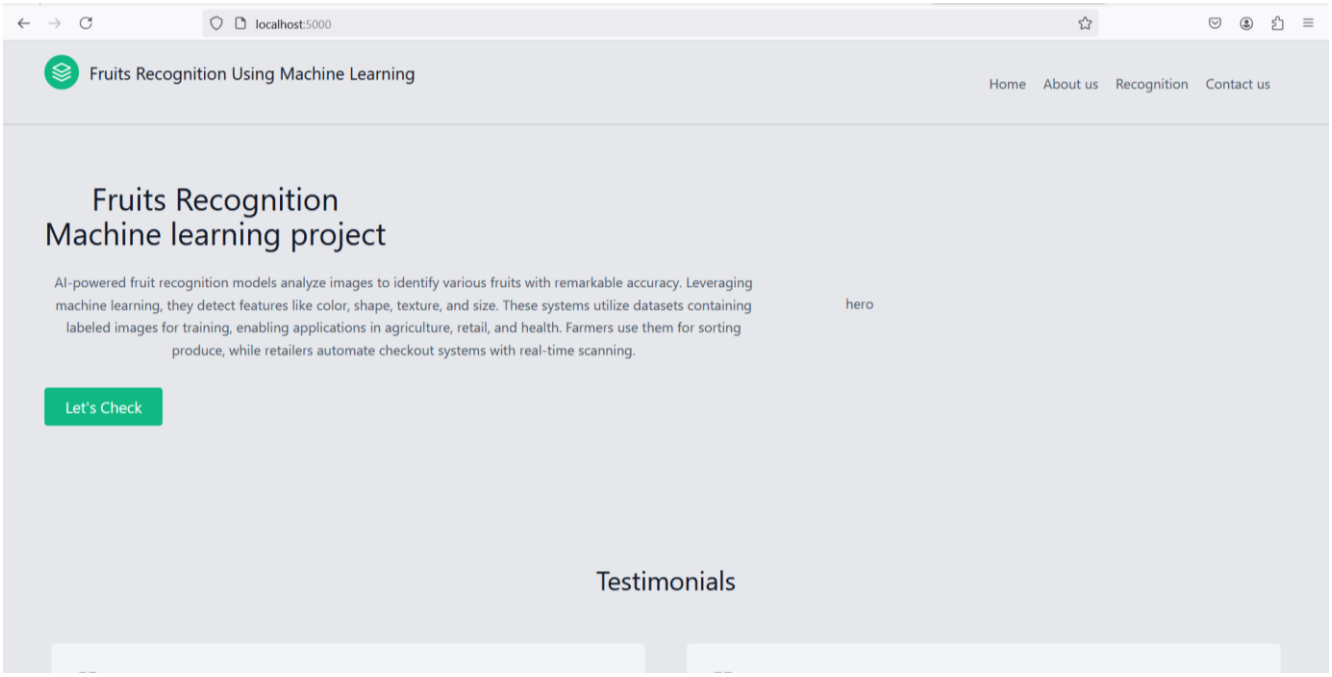
```
Best Random Forest model saved successfully!
```

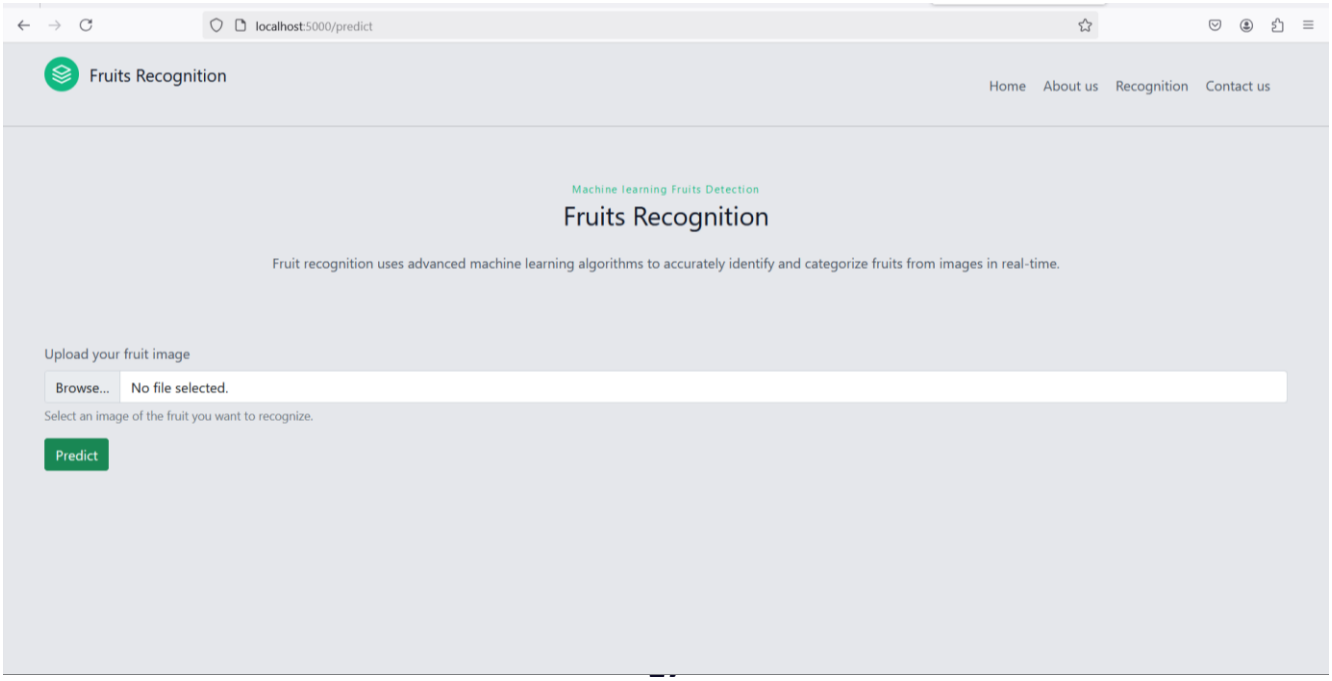
Final Application

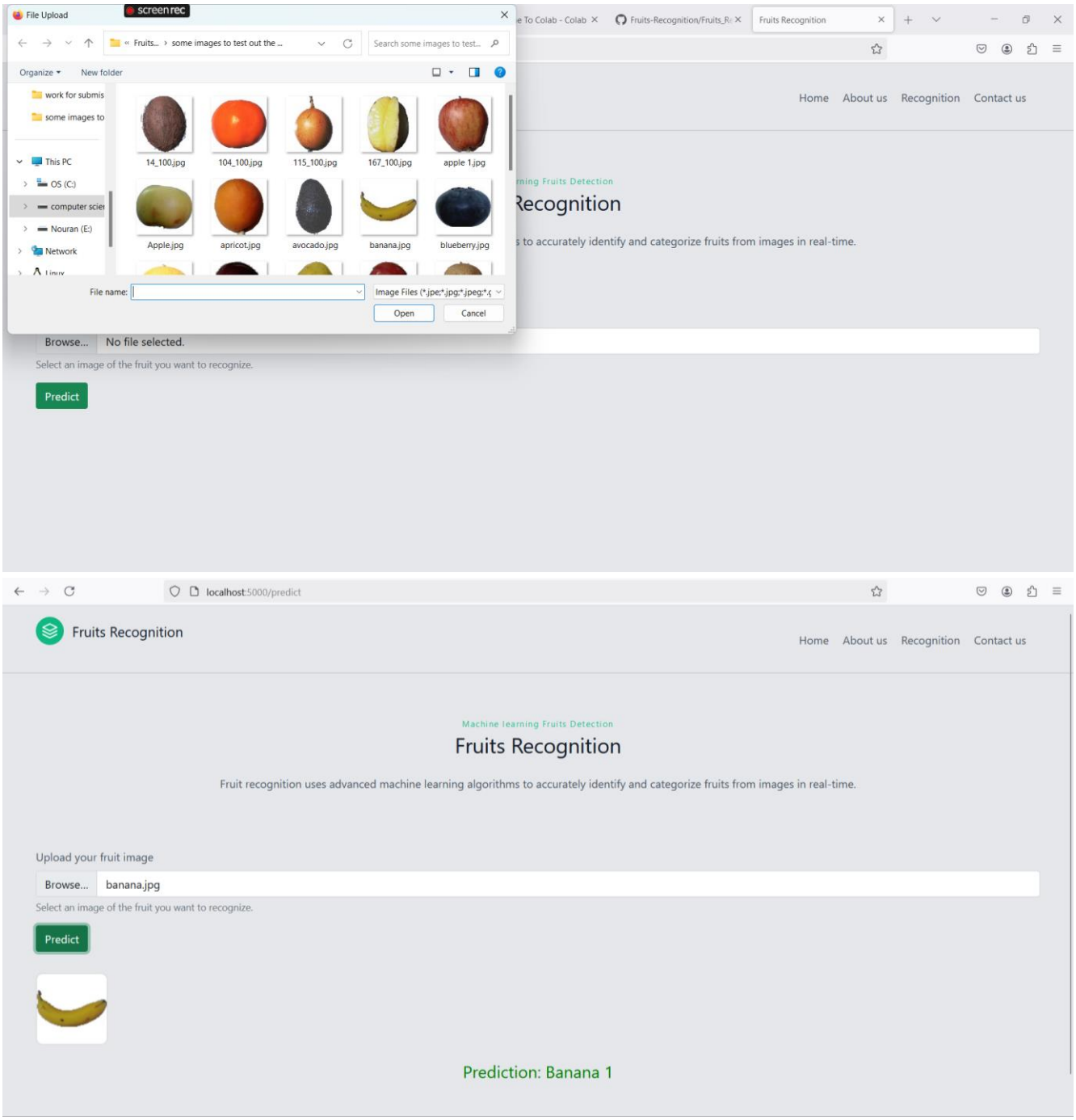
Website Implementation

After training the machine learning model, we have integrated it into a Flask-based web application to provide real-time fruit recognition. The web app allows users to upload fruit images and get predictions based on the trained model.

Photos:






File Upload
screenrec
Fruits Recognition
Home
About us
Recognition
Contact us
Machine learning Fruits Detection
Recognition
to accurately identify and categorize fruits from images in real-time.
Browse...
No file selected.
Select an image of the fruit you want to recognize.
Predict
localhost:5000/predict
Fruits Recognition
Home
About us
Recognition
Contact us
Machine learning Fruits Detection
Fruits Recognition
Fruit recognition uses advanced machine learning algorithms to accurately identify and categorize fruits from images in real-time.
Upload your fruit image
Browse...
banana.jpg
Select an image of the fruit you want to recognize.
Predict
Prediction: Banana 1