Misr International University

Fruits Recognition

Mohamed Hesham – 2023/00428

Mohamed Hesham
Fall 2025

Introduction

Automated fruit recognition represents a significant challenge in computer vision with practical applications spanning automated retail systems, agricultural quality assessment, and dietary monitoring solutions. This project implements a comprehensive fruit identification system leveraging image processing techniques combined with machine learning algorithms. The primary objectives include effective image preprocessing, meaningful feature extraction, robust model training, and deployment of a user-friendly application capable of real-time fruit classification.

Technical Methodology

Dataset

The project utilizes the **Fruits-360 dataset** available on Kaggle:

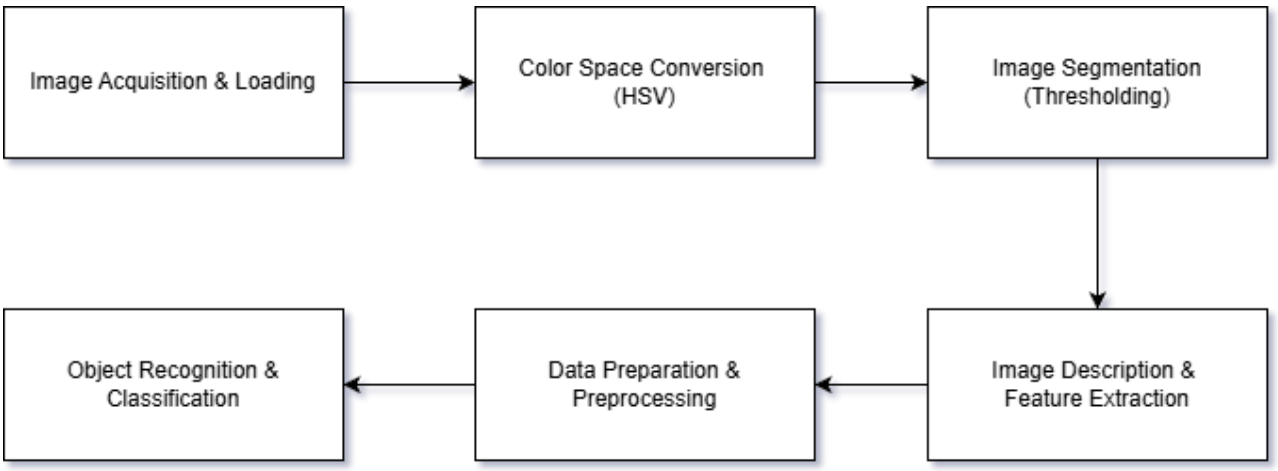
- https://www.kaggle.com/datasets/moltean/fruits

Dataset Overview

The dataset consists of fruit images categorized into distinct classes, organized within separate directories for **training** and **testing** purposes. This structured organization facilitates systematic model development and evaluation. The dataset was selected for its diversity, consistency, and well-defined categorization, making it ideal for developing and validating the recognition model.

Pipeline Diagram

The implementation follows a structured processing pipeline as illustrated below:



Implementation Details

1. Image Acquisition & Loading

The dataset structure consists of training and testing directories, each containing class-specific subfolders. System validation confirms directory existence before processing.

Directory Structure:

- Training path: /dataset/fruits-360/Training
- Testing path: /dataset/fruits-360/Test

Custom functions were developed to load images and corresponding labels, converting images into NumPy arrays for computational processing. A dedicated **Fruit** class was implemented to encapsulate image data and labels, facilitating organized data management

```python
class Fruit:
  # Constructor With a 2 properties: image & label
  def __init__(self, image, label):
      self.image = image
      self.label = label

  # Getter for image
  @property
  def image(self):
      return self._image

  # Setter for image
  @image.setter
  def image(self, image):
      self._image = image

  # Getter for label
  @property
  def label(self):
      return self._label

  # Setter for label
  @label.setter
  def label(self, label):
      self._label = label
```

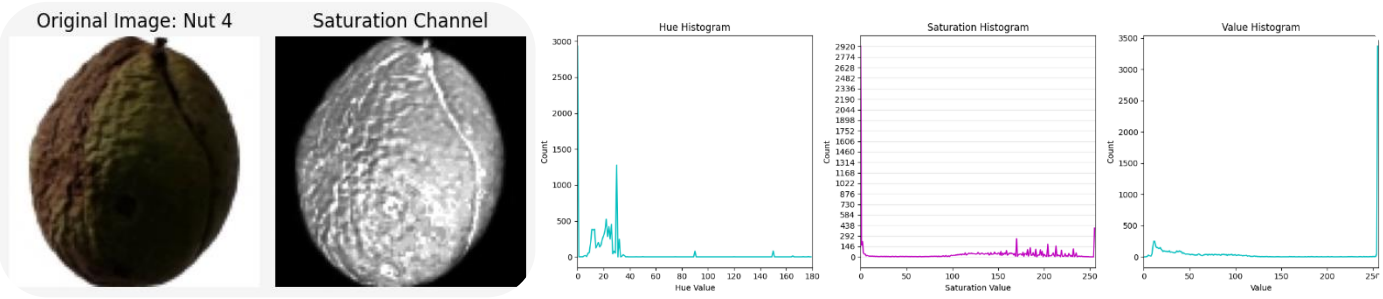
2. Color Space Conversion (HSV)

Images undergo conversion from BGR (OpenCV default) to **HSV** color space. This transformation facilitates more effective segmentation by separating color information from intensity.

HSV Components:
1. **Hue**: Dominant color (0-360° circular scale)
2. **Saturation**: Color purity/intensity
3. **Value**: Brightness component

Conversion Implementation & Visualization

```
# Convert the BGR image to HSV
image_hsv = cv.cvtColor(fruit.image, cv.COLOR_BGR2HSV)
hue, saturation, value = cv.split(image_hsv) # Returns numpy arrays
```



3. Image Segmentation (Thresholding)

Region-based segmentation that isolates fruit regions from the background using **saturation** channel thresholding. The white backgrounds in the dataset have near-zero saturation values, while fruit regions show higher saturation, enabling effective separation.

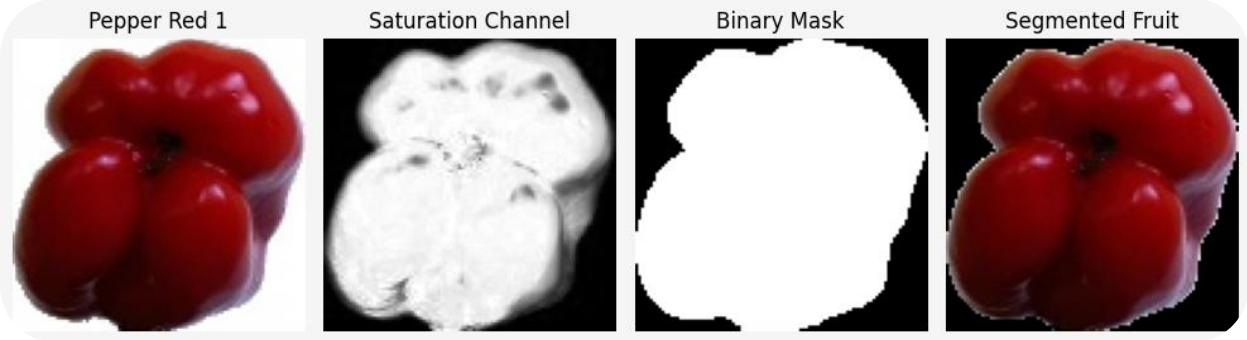
Threshold value: 15 (chosen through trial & error), results in a binary mask where:
- fruit = white (255)
- background = black (0)

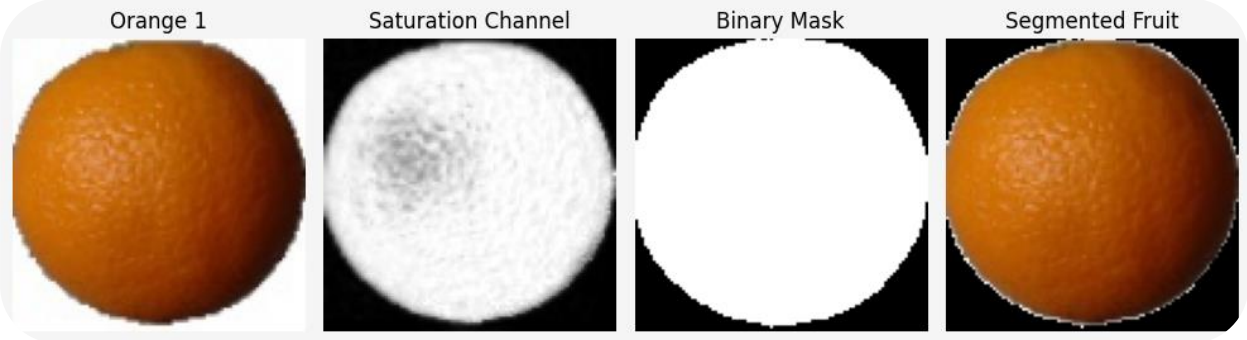
Thresholding Implementation & Visualization:

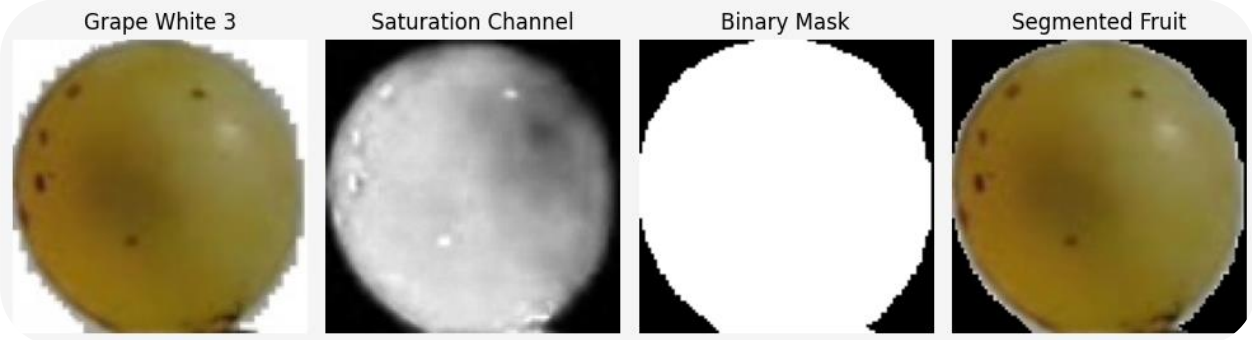
```python
def segment_fruit(fruit):
    # Convert the BGR image to HSV
    image_hsv = cv.cvtColor(fruit.image, cv.COLOR_BGR2HSV)
    hue, saturation, value = cv.split(image_hsv) # Returns numpy arrays

    # Segment the image through simple global thresholding
    _, segmented_mask = cv.threshold(saturation, 15, 255, cv.THRESH_BINARY)

    return [hue, saturation, value, segmented_mask]
```







4. Image Description & Feature Extraction

Four distinctive features were extracted from segmented fruit regions:

1. **Area**: Pixel count of fruit region (region-based descriptor)
2. **Average Hue**: Dominant color using circular mean calculation
3. **Average Saturation**: Mean color purity
4. **Average Value**: Mean brightness intensity.

Feature Extraction Implementation

```python
# This function assumes an already segmented fruit & it's associated channels of HSV
def extract_features(hue, saturation, value, segmented_mask):
    # 1. Area
    area = np.count_nonzero(segmented_mask)

    # Utilize numpy's feature of masking into the arrays immediatly! (Filters True, & removes False values based on the mask!)
    fruit_hue = hue[segmented_mask == 255]
    fruit_saturation = saturation[segmented_mask == 255]
    fruit_value = value[segmented_mask == 255]

    # Average ONLY the fruit pixels
    avg_hue = circmean(fruit_hue * 2, high=360, low=0) / 2 # 2. Average Hue
    avg_saturation = np.mean(fruit_saturation) # 3. Average Saturation
    avg_value = np.mean(fruit_value) # 4. Average Value

    # Return feature vector of that fruit
    return [int(area), float(avg_hue), float(avg_saturation), float(avg_value)]
```

Process All Fruits (Segmentation & Feature Extraction)

```python
def process_all_fruits(fruits):
    # Define a list for all features & another for all labels (both correspond in index)
    all_features = []
    all_labels = []

    for fruit in fruits:
        # Segment the fruit
        hue, saturation, value, mask = segment_fruit(fruit)

        # Extract features from the segmented fruit
        feature_vector = extract_features(hue, saturation, value, mask)

        # Store the features & labels in their respective lists
        all_features.append(feature_vector)
        all_labels.append(fruit.label)

    # Create DataFrame with proper column names
    df = pd.DataFrame(
        data=all_features,
        columns=['Area', 'Average Hue', 'Average Saturation', 'Average Value']
    )

    # Add label/class column - associated with a label because it's training data
    df['Label'] = all_labels

    # Return the training data!
    return df
```

	Area	Average Hue	Average Saturation	Average Value	Label
0	6834	12.793744	144.172081	141.785777	Apple 10
1	6524	7.317757	179.534795	123.980687	Apple 10
2	6496	7.376361	176.911792	125.764624	Apple 10
3	7977	12.326224	136.595462	140.741758	Apple 10
4	6435	7.636385	176.527584	126.196426	Apple 10
...	...	...	...	...	...
121106	1733	55.033976	50.454126	52.240623	Zucchini dark 1
121107	1996	58.476406	42.183868	53.874248	Zucchini dark 1
121108	1869	57.797903	42.788122	50.217228	Zucchini dark 1
121109	1648	54.659064	46.624393	46.340413	Zucchini dark 1
121110	1754	57.792817	40.533637	50.465792	Zucchini dark 1

121111 rows × 5 columns

5. Data Preparation and Preprocessing

Extracted features undergo **standardization** using Min-Max scaling to ensure uniform value ranges (0-1), optimizing them for machine learning algorithms. Label **encoding** transforms categorical fruit names into numerical representations.

Preprocessing Steps:

1. Feature normalization using **MinMaxScaler**
2. Label encoding using **LabelEncoder**
3. Data splitting into training and testing sets.

Preprocessing Implementation

```
# Separate features and labels (because only the labels are the string one)
X_train = fruit_df[['Area', 'Average Hue', 'Average Saturation', 'Average Value']]
y_train = fruit_df['Label']

# Normalize features from 0 to 1 (better for the algorithms)
minmax_scaler = MinMaxScaler()
X_train_normalized = minmax_scaler.fit_transform(X_train)  # numpy array

# Encode string labels to numerical (required for the algorithms to work)
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)  # numpy array
```

Preprocessing Results (Before vs. After)

```
Before Normalization (0 to 1 range)
------------------------------------------------------------
   Area  Average Hue  Average Saturation  Average Value
0  6834    12.793744          144.172081     141.785777
1  6524     7.317757          179.534795     123.980687
2  6496     7.376361          176.911792     125.764624
3  7977    12.326224          136.595462     140.741758
4  6435     7.636385          176.527584     126.196426

After Normalization (0 to 1 range)
------------------------------------------------------------
[[0.78512293 0.07108682 0.58138645 0.54937618]
 [0.74623683 0.04065576 0.7499613  0.46881504]
 [0.74272454 0.04098143 0.73745739 0.47688667]
 [0.92849975 0.06848872 0.54526853 0.5446524 ]
 [0.73507275 0.04242644 0.73562586 0.4788404 ]]

Before Encoding String Labels To Numerical
------------------------------------------------------------
['Apple 10' 'Apple 11' 'Apple 12' 'Apple 13' 'Apple 14' 'Apple 17'
 'Apple 18' 'Apple 19' 'Apple 5' 'Apple 6' 'Apple 7' 'Apple 8' 'Apple 9'
 'Apple Braeburn 1' 'Apple Core 1' 'Apple Crimson Snow 1' 'Apple Golden 1'
 'Apple Golden 2' 'Apple Golden 3' 'Apple Granny Smith 1']

After Encoding String Labels To Numerical
------------------------------------------------------------
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

6. Object Recognition & Classification

Model Training and Selection

Multiple machine learning algorithms were evaluated:

- K-Nearest Neighbors (**KNN**): Implemented with Euclidean distance metric
- **Random Forest**: Ensemble method with multiple decision trees

Training & Testing Results

Models were trained on normalized feature vectors with encoded labels, employing separate training and testing datasets.

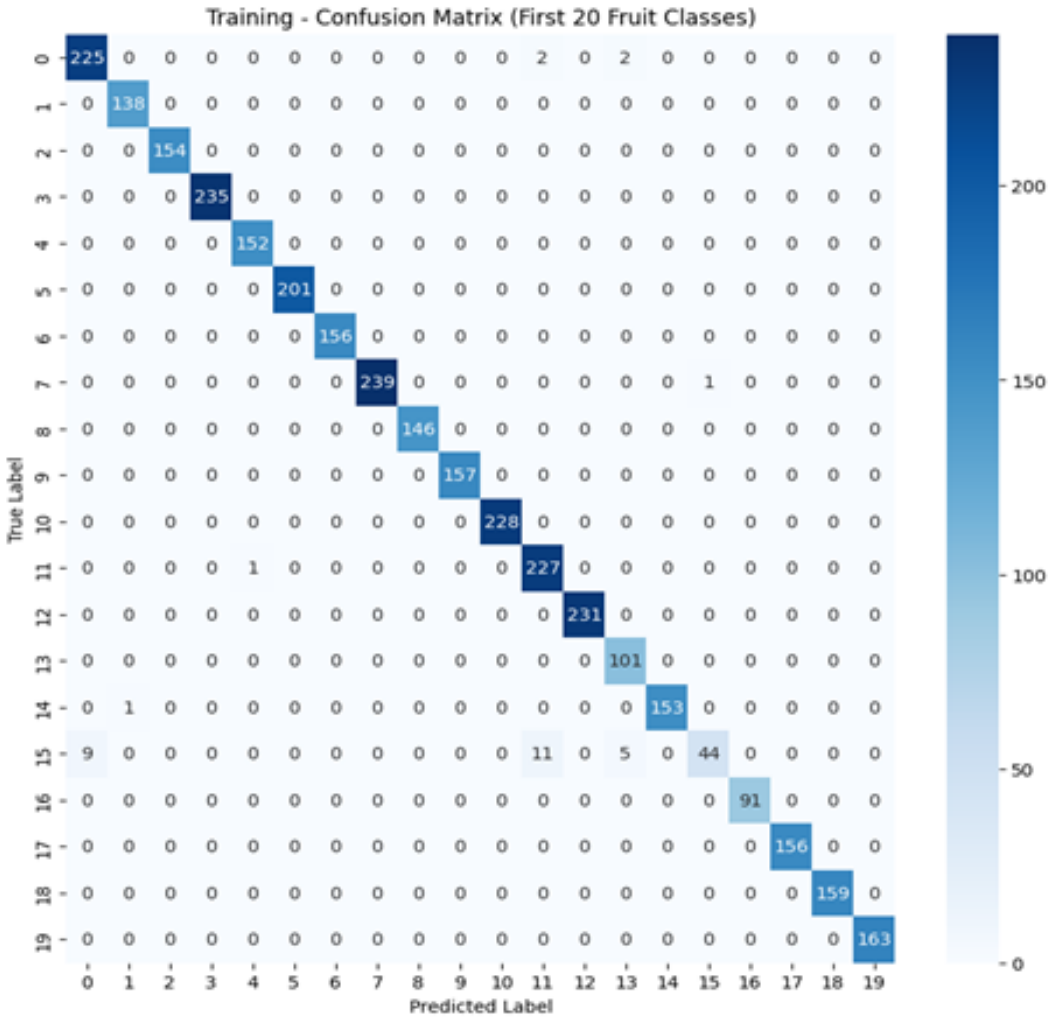
Random Forest achieved **92.04%** accuracy on the **testing** data. Thus, the model `*fruit_model.joblib*` is saved with its scaler & encoder for deployment (website implementation).

Visualization Of Predictions

Confusion Matrix Visualization & Implementation

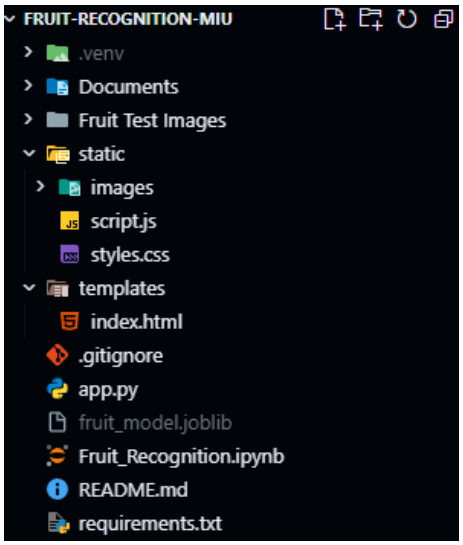
```python
def visualize_confusion_matrix(labels, predicted_labels):
    # Create confusion matrix
    cm = confusion_matrix(labels, predicted_labels)

    # Show a smaller version (first 20 classes) so it's readable
    plt.figure(figsize=(10, 10))
    sns.heatmap(cm[:20, :20], annot=True, fmt='d', cmap='Blues') # annot: shows numbers in heatmap, fmt: decimal
    plt.title('Training - Confusion Matrix (First 20 Fruit Classes)')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()
```



Web Application Deployment

Flask-Based Implementation

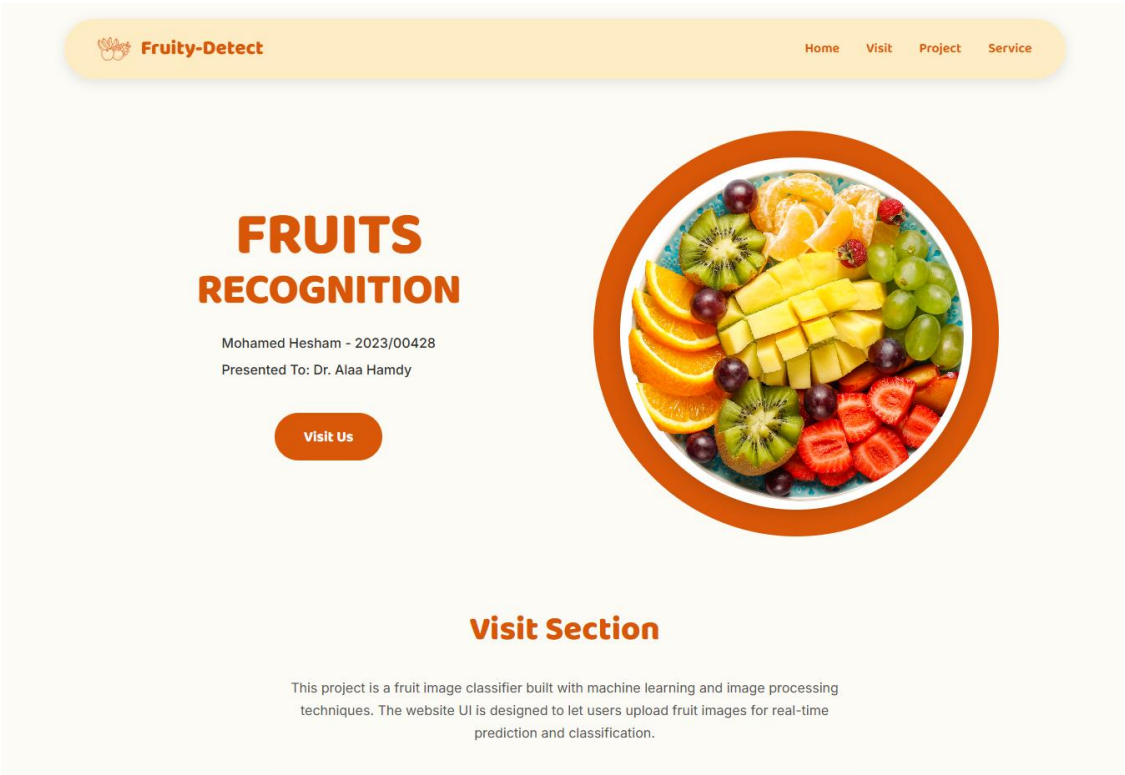


The trained model was integrated into a **Flask** web application providing real-time fruit recognition capabilities.

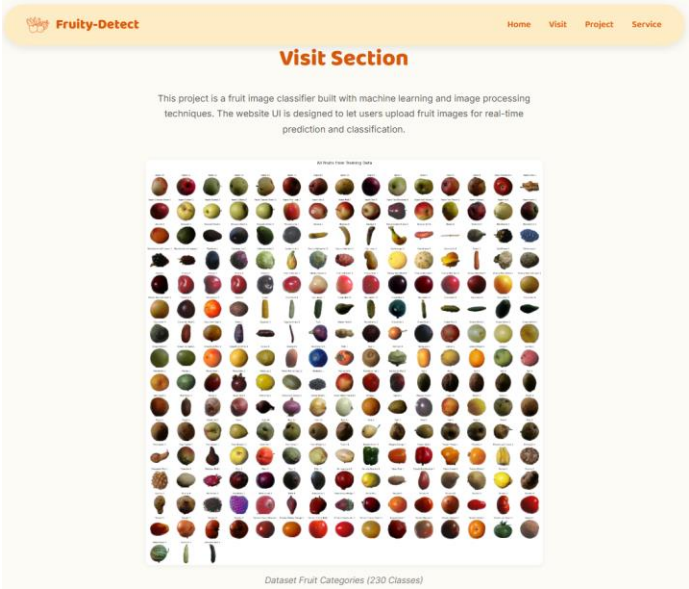
Application Features:

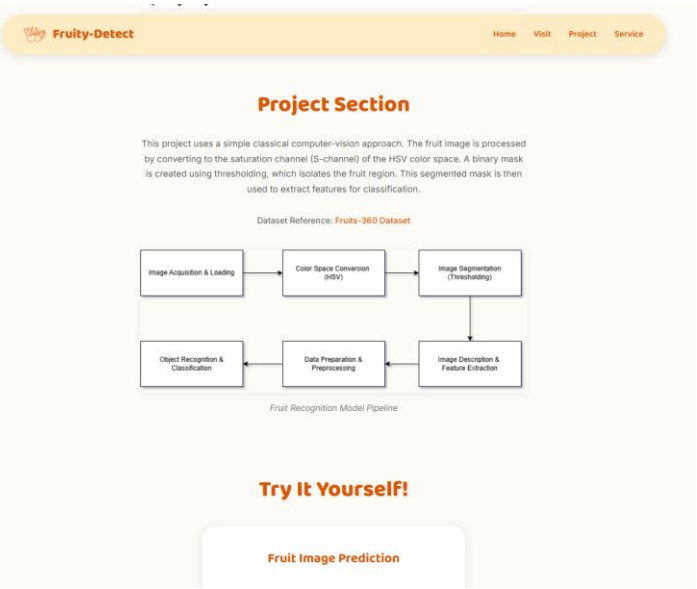
- User-friendly image upload interface
- Real-time prediction with confidence scoring
- Responsive design for various devices
- Error handling for invalid inputs

Home Section



Visit & Project Sections





Prediction Section (Try it yourself!)