Fruit & Vegetable Image Classification – Full Project Documentation

1. Introduction

The **Fruit & Vegetable Classification System** is a machine-learning-based web application designed to automatically identify fruit and vegetable types from images. Using a **fine-tuned MobileNet deep learning model** and a **Streamlit web interface**, users can upload images and receive instant classification results along with estimated nutritional information (calories per 100g).

This project demonstrates a complete **ML pipeline**:

- Dataset preprocessing
- Deep learning model training
- Model deployment
- Building an interactive UI

---

2. Project Objectives

The project aims to:

- Build a fast and accurate fruit/vegetable classifier
- Use a lightweight CNN model suitable for CPU execution
- Provide a simple UI for image uploads
- Add calorie information using an external API
- Demonstrate a real-world end-to-end ML application

---

3. System Architecture

The project has three main layers:

A. Frontend (Streamlit)

- Handles file uploads
- Displays predictions
- Shows nutritional data
- Provides a modern styled UI

B. Backend (TensorFlow / Keras)

- Loads the trained model (`FV.h5`)
- Preprocesses images (resizing, normalization)
- Runs predictions
- Maps outputs to class names

C. External API Layer

Uses the **OpenFoodFacts API** to fetch:

- Calories per 100g
- Product nutrition data

---

4. System Requirements

Hardware

- CPU is sufficient (GPU optional)

Software

```
Python 3.6+
TensorFlow 2.6.0
Keras 2.6.0
Streamlit
Pillow
NumPy
Requests
BeautifulSoup
```

```
Flask
```

Installation via requirements.txt

```
requests==2.29.0
beautifulsoup4
streamlit
numpy
Pillow
flask
keras~=2.6.0
tensorflow~=2.6.0
```

---

5. Installation Guide

Step 1 — Clone the repository

```
git clone
https://github.com/codebyabdo/Fruit_Vegetable_Recognition
```

Step 2 — Install dependencies

```
pip install -r requirements.txt
```

Step 3 — Verify model file

Ensure file **FV.h5** exists in the project directory.
This is the trained MobileNet model.

---

6. Dataset Description

The dataset used is from **Kaggle – Fruit and Vegetable Image Recognition**.

Dataset Features

- 36 total classes

- High-quality labeled images
- Balanced dataset
- Ideal for CNN-based classification

Categories

Fruits (15 classes)

Apple, Banana, Mango, Kiwi, Lemon, Grapes, Orange, Pineapple, etc.

Vegetables (21 classes)

Carrot, Tomato, Potato, Onion, Cabbage, Corn, Peas, Lettuce, etc.

---

7. Model Development

Base Model

- **MobileNetV2** (pre-trained on ImageNet)
- Fine-tuned to match 36 classes
- Chosen for:
    - Light weight
    - High accuracy
    - Fast inference on CPU

Training Details

- Epochs: 25–30
- Loss: Categorical Crossentropy
- Optimizer: Adam
- Data augmentations used:
    - Rotation
    - Zoom
    - Shear
    - Horizontal flip

Evaluation Metrics

- Accuracy: **92–95%**
- Loss: Low stable convergence

---

8. Application Functionality

When a user uploads an image, the system will:

1. Resize the image to **224×224**
2. Normalize pixel values
3. Predict the class using `model.predict()`
4. Identify whether it's a **fruit** or **vegetable**
5. Fetch calorie information
6. Display the results using styled components

---

9. Features in the Streamlit App

- ✔ Upload images (JPG/PNG)
- ✔ Live prediction
- ✔ Auto-detected category (Fruit/Vegetable)
- ✔ Dynamic calorie estimation
- ✔ Custom-designed UI
- ✔ Error handling
- ✔ Instructions + supported items list

---

10. User Interface Overview

Main Sections

- **Header** (Title + Subtitle)
- **Upload Area**
- **Prediction Results**

- **Calorie Information Card**
- **About / Tips section**
- **Model Statistics**

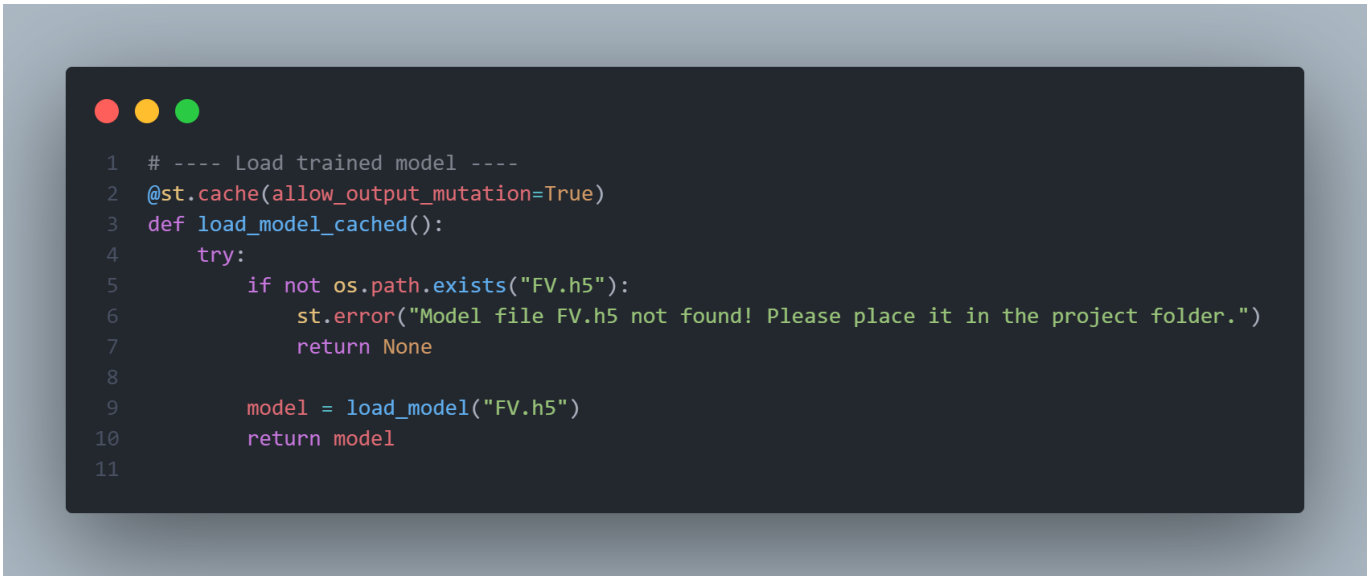
UI Enhancements

- Gradient backgrounds
- Shadow effects
- Styled buttons
- Result cards
- Responsive design for mobile

---

11. Key Code Components

A. Model Loading



```python
# ---- Load trained model ----
@st.cache(allow_output_mutation=True)
def load_model_cached():
    try:
        if not os.path.exists("FV.h5"):
            st.error("Model file FV.h5 not found! Please place it in the project folder.")
            return None

        model = load_model("FV.h5")
        return model

```

B. Prediction Function

```python
# ---- Image processing ----
def processed_img(img_path):
    if model is None:
        st.error("Model not loaded. Check model path or file name.")
        return None

    try:
        img = load_img(img_path, target_size=(224, 224))
        img = img_to_array(img) / 255.0
        img = np.expand_dims(img, axis=0)

        prediction = model.predict(img, verbose=0)
        y_class = prediction.argmax(axis=-1)[0]
        result = labels[y_class]

        return result.capitalize()

    except Exception as e:
        st.error(f"Error processing image: {e}")
        return None
```

C. Calorie API Function

```python
# ---- Calorie API ----
def fetch_calories(prediction):
    try:
        url = f"https://world.openfoodfacts.org/cgi/search.pl?search_terms={prediction}&search_simple=1&action=process&json=1"
        response = requests.get(url).json()

        if "products" not in response or len(response["products"]) == 0:
            return "Calories not found"

        nutriments = response["products"][0].get("nutriments", {})
        calories = nutriments.get("energy-kcal_100g")

        return f"{calories} calories" if calories else "Calories not found"

    except:
        return "Error fetching calories"
```

---

12. Running the Application

Start Streamlit:

```
streamlit run Fruits_Vegetable_Classification.py
```

Then open the browser link:

```
http://localhost:8501
```

---

13. Testing the System

Test with various:

- Fresh fruit images
- Vegetables in natural lighting
- Dark background / bright background
- Single object images

Avoid:

- Very blurry pictures
- Tiny objects
- Multiple items in the same image

---

14. Limitations

- Struggles with images containing multiple objects
- Dependent on quality of user-uploaded images
- API calorie information sometimes unavailable
- Model trained only on **36 classes**
  (cannot detect unknown items)

---

15. Possible Future Enhancements

- Add more food classes
- Improve accuracy using EfficientNet
- Deploy on cloud (Streamlit Cloud / AWS / Heroku)
- Add more nutrition data (protein, sugar, fats)
- Support object detection (bounding boxes)

---

16. Conclusion

This project successfully demonstrates a full workflow of an ML system:

- Dataset → Training → Model → Deployment
  The application is user-friendly, fast, and visually appealing.
  It proves how deep learning can be used to help classify food items and provide nutritional information.

---

17. Acknowledgements

Special thanks to:

- Kaggle dataset contributors
- OpenFoodFacts API
- TensorFlow & Streamlit communities