Real-Time Image Classifier

1. Project Goal and Overview

The primary objective of this project was to develop a robust deep learning model capable of classifying images of **fruits and vegetables** and to deploy this model as a user-friendly, real-time web application using Streamlit.

- **Dataset:** Custom **Fruits_Vegetables** dataset (containing **36 classes**).
- **Model Type:** Custom **Convolutional Neural Network (CNN)**.
- **Deployment:** Streamlit Dashboard for real-time inference.

2. Data Preparation and Preprocessing

The data pipeline was crucial for ensuring the model's accuracy and ability to generalize to new, unseen images.

A. Data Splitting and Loading

The dataset was organized into three distinct directories to create the necessary splits: `train`, `validation`, and `test`.

- **Loading Utility:** `tf.keras.utils.image_dataset_from_directory` was used to load images directly from the folder structure.
- **Parameters:**
  - `image_size`: All images were resized to a consistent size (e.g., 180*180 pixels).
  - `batch_size`: 32
  - **Labels:** Automatically inferred as integer labels (sparse encoding).

B. Normalization and Rescaling

Normalization is applied to standardize pixel values, which is necessary for optimal model training.

- **Method:** A dedicated **`layers.Rescaling(1./255)`** layer was used.
- **Function:** This scales the raw pixel values (which range from 0 to 255) to a floating-point range of **[0, 1]**

C. Image Augmentation

To prevent overfitting and increase the diversity of the training data, an aggressive augmentation pipeline was implemented **only on the training set**.

Augmentation Layer	Function
`RandomFlip`	Randomly flips images horizontally or vertically.
`RandomRotation(0.2)`	Randomly rotates images by up to ±36°

3. Model Development and Training

A custom CNN was designed and trained to handle the complexity of the 36-class image classification task.

A. Algorithm Selection and Architecture

A custom CNN was chosen to extract hierarchical features from the image data.

- **Structure:** Sequential model consisting of multiple blocks of Conv2D→ MaxPooling2
- **Regularization: Dropout layers** (e.g., 0.5) were included before the final dense layers to mitigate overfitting.
- **Output Layer:** The final layer has **36 units** with a **`softmax`** activation function, providing a probability distribution over the 36 classes.

B. Compilation and Training

The model was compiled and trained using standard, optimized settings.

- **Optimizer:** **`Adam`** (adaptive learning rate optimization).
- **Loss Function:** **`sparse_categorical_crossentropy`** (appropriate for integer-encoded labels).
- **Early Stopping:** A callback was used to monitor the validation loss and stop training if the loss did not improve after a set number of epochs (patience), saving the best performing weights.

4. Evaluation and Prediction

A. Model Evaluation

The final model was evaluated on the unseen **test set** to determine its real-world performance.

- **Primary Metric:** Accuracy (tracked throughout training and final evaluation).
- **Detailed Metrics:** Code was executed to generate a **Classification Report** (Precision, Recall, F1-Score) and a **Confusion Matrix**, providing a detailed view of class-specific performance.

B. Real-Time Prediction

The notebook includes utility code demonstrating the prediction workflow for a single image, which forms the basis for the Streamlit application:

1. Load and display the image.
2. Preprocess the image (resizing, normalization).
3. Expand dimensions to create a batch size of 1 (`tf.expand_dims`).
4. Call `model.predict()`.
5. Apply `tf.nn.softmax` to the output to get class probabilities.
6. Use `tf.argmax` to select the highest-scoring class index.
7. Display the predicted class name and confidence score

5. Deployment (Streamlit)

The final step involves using the trained model (`image_classifier_model.keras`) within a Streamlit application (`app.py`) to create an interactive web interface for real-time prediction. This allows users to upload their own images and receive instant, visualized classification results. Below are some snapshots of the deliverables:

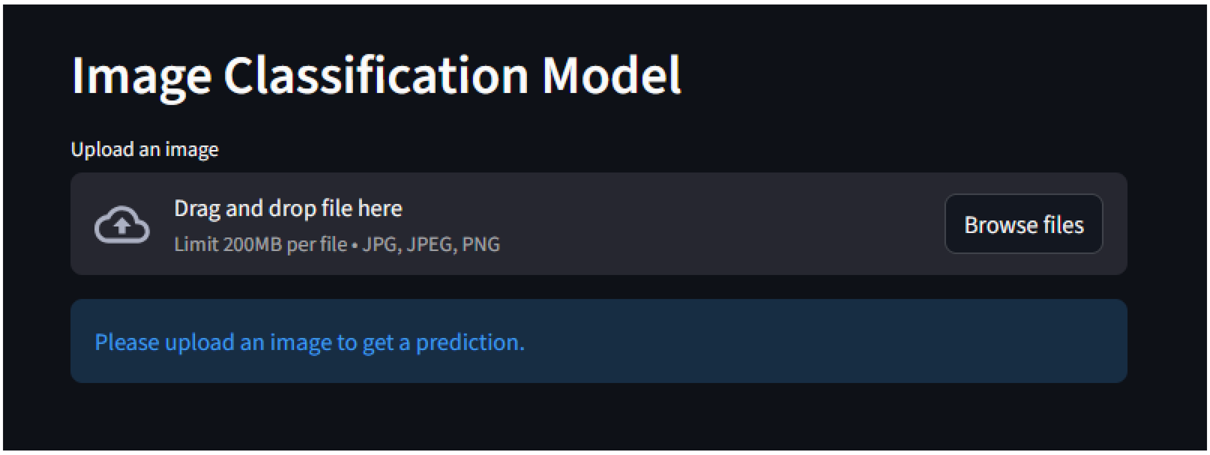

Figure 1: Image uploader

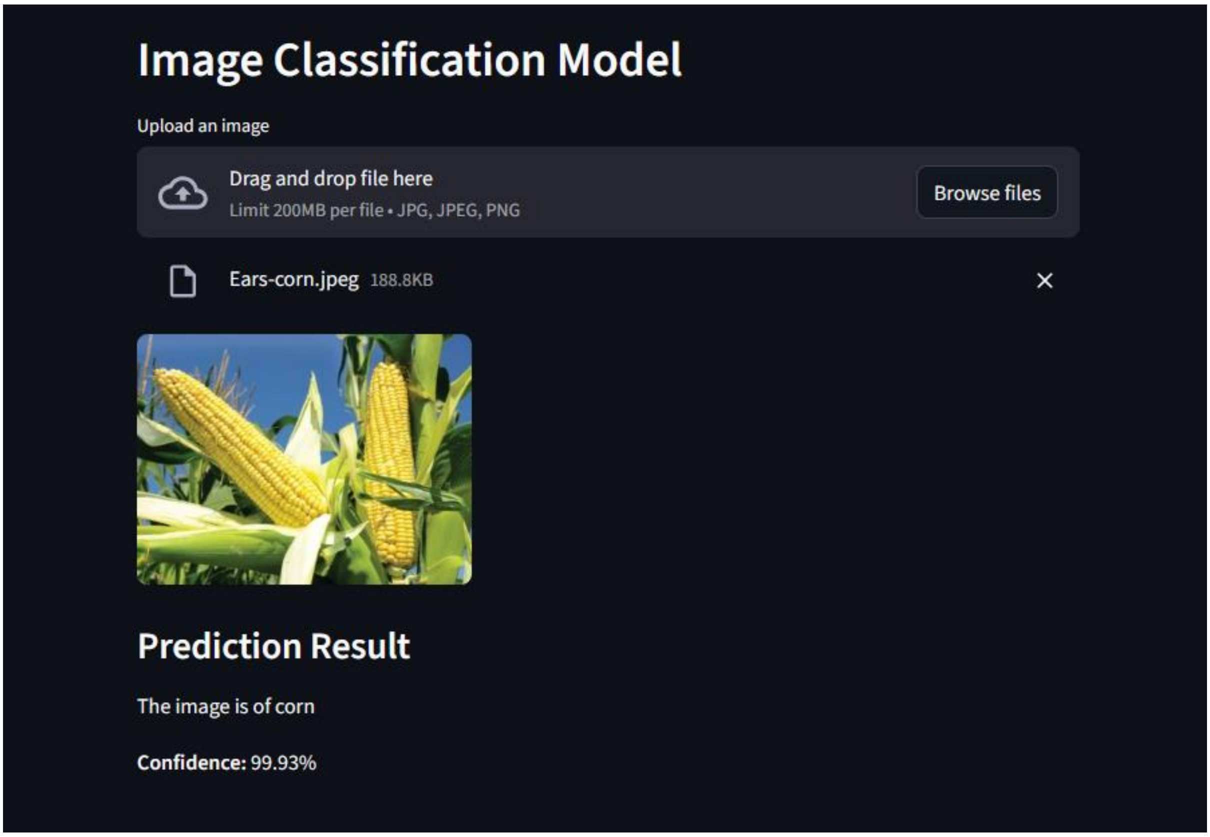

Figure 2: Classification Example