Fruit & Vegetable Classification

1. System Input And Output

1.1 System Input

- The primary input is a single image file uploaded by the user. The system supports a wide range of standard image formats (e.g., jpg, jpeg, png, webp, gif). For animated formats like gif, the system extracts and processes only the first frame

- The system can also accept a live video feed captured directly from a standard user-facing webcam, processing the feed on a frame-by-frame basis

1.2 System Output

- **Categorical Classification**: For a static image input, the output is a prediction with:
  + Class Label: A string identifying the object from 36 possible classes (e.g., “Apple”, “Banana”)
  + Confidence Score: A percentage representing the model’s certainty

- **Interpretability Visualization (Grad-CAM)**: A secondary output for static images, the system generates a Grad-CAM (Gradient-weighted Class Activation Mapping) heatmap. This visualization is overlaid on the original image to highlight the specific pixel regions the model used to make its classification, offering insight into the model’s decision-making process

- **Annotated Video Stream**: For a real-time video input, the output is a modified video stream. The system overlays the top-ranked class label and confidence score directly onto each frame, providing a continuous, real-time classification of the objects in view.

2. Obtained Outcomes

2.1 Model Performance and Comparison

- Two distinct Convolutional Neural Network (CNN) architectures, MobileNetV2 and EfficientNetV2-B0, were trained using transfer learning. The pre-trained base layers of each model were frozen, and a custom classification head was trained on the 36-class fruit and vegetable dataset.

- Both models were trained under similar conditions (transfer learning, 50-epoch limit, EarlyStopping with patience=6) and then evaluated on the same withheld test dataset. The MobileNetV2 model had no dropout regularization, while the EfficientNetV2-B0 model included two 30% Dropout layers.
- The final performance metrics and evaluation times on the test set are as follows:

Model	Test Accuracy	Test Loss	Evaluation Time
MobileNetV2	94.01%	0.1501	157 seconds
EfficientNetV2-B0	92.81%	0.1794	31 seconds

-

2.2 Deployed Application

- A multi-page web application was developed using Streamlit to serve as an user-friendly interface for the trained models. The application is modular and provides four core features accessible from a navigation sidebar:

    1. Run Prediction: A page where users can upload a single image file. The interface provides a dropdown menu to select either the MobileNetV2 or EfficientNetV2-B0 model. The application then displays the top-ranked prediction and its corresponding confidence score

    2. Grad-CAM: A tool that implements Gradient-weighted Class Activation Mapping (Grad-CAM). Users can upload an image and select a model to generate a visual heatmap, which highlights the specific pixel regions the model utilized to make its classification

    3. Real Time Demo: A module that uses streamlit-webrtc to access the user's webcam. It performs inference on each frame of the live video feed, overlaying the top-ranked prediction and confidence score in real-time
    .
    4. Notebooks: A page that embeds and displays the original Colab notebooks used for model training and evaluation.

3. Project Source Code

- The complete source code for this project, including the training notebooks, evaluation notebooks, and the final application, is publicly available on GitHub: https://github.com/BSFactor/Fruit-Classifier

4. Data Source

- This project was trained on the Fruit and Vegetable Image Recognition dataset from Kaggle by Kritik Seth.

- Dataset Link: https://www.kaggle.com/datasets/kritikseth/fruit-and-vegetable-image-recognition

5. Project Workflow

1. **Data Preparation**: The dataset was acquired from Kaggle using the kagglehub library. A custom function (*proc_img*) ingested the data by parsing directory structures to extract class labels from folder names and mapping them to their corresponding filepaths. The resulting paths and labels were compiled into train, validation, and test Pandas DataFrames.

2. **Data Pipelines**: *keras.preprocessing.image.ImageDataGenerator* was used to create separate data pipelines for each model. The training pipeline applied on-the-fly data augmentation (e.g., rotation, zoom) for regularization. Each pipeline was instantiated with its model-specific preprocessing function (*mobilenet_v2.preprocess_input* or *efficientnet_v2.preprocess_input*) to ensure correct input normalization.

3. **Model Architecture**: The project employed transfer learning. Both MobileNetV2 and EfficientNetV2-B0 were loaded with pre-trained ImageNet weights, and their top classification layers were removed (*include_top=False*). The bases were frozen (*trainable = False*). A new classification head—consisting of *Dense* layers and *Dropout* (for EfficientNet)—was appended, terminating in a 36-unit softmax output layer.

4. **Model Training**: Models were trained in a Google Colab environment. To prevent overfitting, the EarlyStopping callback was implemented to monitor *val_loss* with a *patience* of 6. The *restore_best_weights=True* parameter ensured that the final serialized model retained the weights from the epoch with the lowest validation loss.

5. **Model Evaluation**: Post-training, the saved models were loaded into an evaluation notebook to evaluate on a test set. Performance was measured using *model.evaluate()*, and error analysis was conducted using confusion matrices and *model.predict()*.

6. **Application Deployment**: The final models were deployed in a multi-page Streamlit web application, which was structured to provide all project features (prediction, visualization, and real-time) in a single, user-friendly interface.

6. Tutorial To Run The Source Code

Step 1: Clone the Repository

Clone the project repository from GitHub to a local directory

```
git clone [Repo URL]
cd [Repo Folder]
```

Step 2: Create and activate a Virtual Environment

```
python -m venv venv
.\venv\Scripts\activate
```

Step 3: Install Dependencies

```
pip install -r requirements.txt
git lfs pull
```

Step 4: Launch Application

Once the environment is activated and all dependencies are installed, the Streamlit web application can be launched

```
streamlit run app.py
```