



Deepfruitveg Automated Fruit And Veg Identification

Project Hand-out, Faculty Development Program

SmartInternz
www.smartinternz.com

Deepfruitveg is an innovative automated system utilizing advanced deep learning algorithms to identify various fruits and vegetables. By analyzing visual characteristics from images, this technology streamlines the identification process, benefiting industries like agriculture, food processing, and retail. With its ability to accurately classify produce, Deepfruitveg enhances efficiency and quality control in diverse settings.

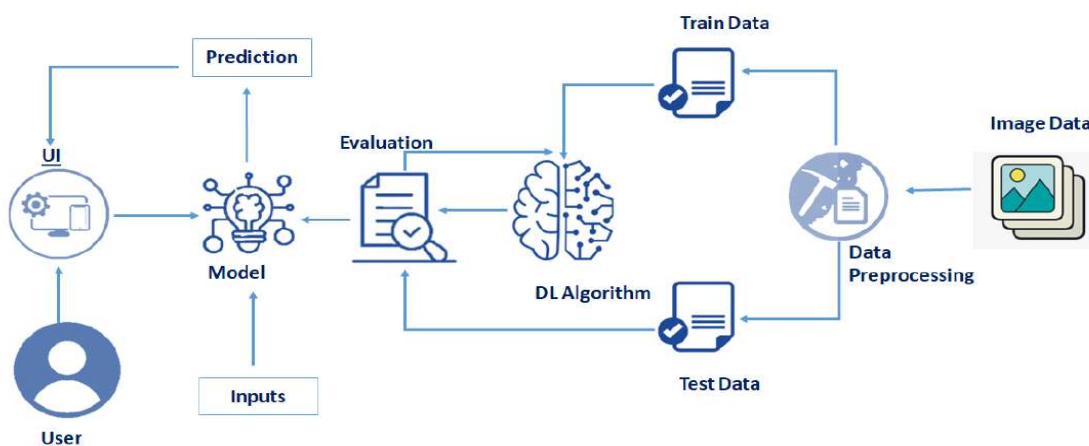
Scenarios:

Automated Sorting in Food Processing Plants: Food processing plants integrate the developed deep learning model into their sorting systems. As fruits and vegetables pass through conveyors, images are captured and analyzed by the model, enabling automated sorting based on visual characteristics. This streamlines the sorting process, improving efficiency and consistency in product quality.

Quality Control in Supermarkets: Supermarkets use the deep learning model to ensure the quality and freshness of fruits and vegetables on their shelves. By conducting periodic inspections using smartphones or dedicated devices, store personnel can quickly verify the classification of produce, reducing the risk of selling inferior products to customers.

Precision Agriculture for Crop Monitoring: Farmers employ the deep learning model for crop monitoring in precision agriculture. Equipped with drones or ground-based cameras, the model analyzes images of crops in real-time, identifying signs of disease or stress based on leaf characteristics. This enables timely intervention, such as targeted pesticide application or irrigation adjustments, to optimize crop health and yield.

Technical Architecture:



Project Flow

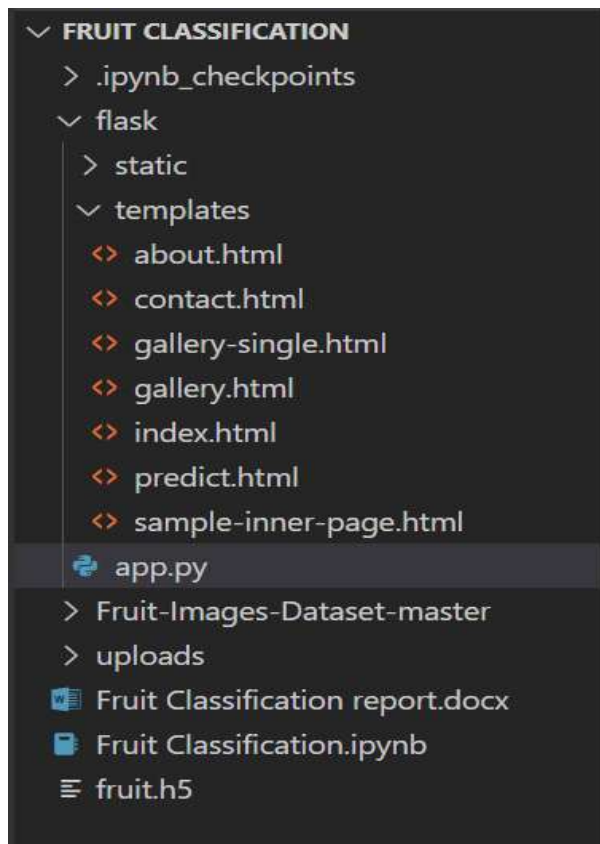
- User interacts with the UI (User Interface) to upload the image as input
- Uploaded image is analyzed by the model which is integrated
- Once model analyzes the uploaded image, the prediction food recipe is showcased on the UI

To accomplish this, we have to complete all the activities and tasks listed below-

- Data Collection
 - Collect the dataset or create the dataset
- Data Preprocessing.
 - Import the ImageDataGenerator library
 - Configure ImageDataGenerator class
 - Apply ImageDataGenerator functionality to train set and test set
- Model Building
 - Import the model building Libraries
 - Initializing the model
 - Create a model using transfer learning with EfficientNetB3
 - Create a custom Keras callback to continue and optionally set LR or halt training
 - Instantiate custom callback
 - Configure the Learning Process
 - Training the model
 - Predictions on the test set ¶
 - Save the Model
 - Create Plot of % change in validation loss for each epoch
- Application Building
 - Create an HTML file
 - Build Python Code

Project Structure

Create a Project folder which contains files as shown below-



- We are building a Flask Application which needs HTML pages stored in the templates folder and a python script app.py for server-side scripting.
- we need the model which is saved and the saved model in this content is grapevine.h5
- The static folder will contain js and css files.

Pre requisites

To complete this project, you must require following software's , concepts and packages

Anaconda navigator:

- Refer to the link below to download anaconda navigator
- Link : <https://www.youtube.com/watch?v=5mDYijMfSzs>

Python packages:

- open anaconda prompt as administrator
- Type "pip install tensorflow" (make sure you are working on python 64 bit)
- Type "pip install flask".

Deep Learning Concepts

- CNN: <https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add>
- Flask Basics : https://www.youtube.com/watch?v=Ij4L_CvBnt0

1. Introduction

1.1 Project Overview

The project titled "**Deepfruitveg: Automated Fruit And Veg Identification**" focuses on developing an AI-powered image classification system. It leverages deep learning and computer vision to automate the identification and classification of various fruits and vegetables. This helps reduce human effort, increase consistency, and streamline processes in the agricultural and food supply chain industries.

1.2 Objectives

- Automate the classification of fruits and vegetables using a robust deep learning model.
- Achieve high classification accuracy across varying image conditions (e.g., lighting, background).
- Deploy the model to assist quality control in food processing plants and supermarkets.
- Make the model scalable and integration-ready for real-time systems.

2. Project Initialization and Planning Phase

2.1 Define Problem Statement

Manual fruit and vegetable classification is time-consuming and prone to human error. Industries such as agriculture and food supply chains need a scalable solution to automate this process efficiently. Variability in background, lighting, and quality further complicates the task.

Two problem personas were identified:

- **PS-1:** A quality control manager needs faster, more accurate sorting.
- **PS-2:** A farm supervisor wants early detection of crop issues without relying on human experts.

2.2 Project Proposal (Proposed Solution)

Use deep learning (CNNs) to build a model that classifies fruits and vegetables based on images. This involves data collection, preprocessing, model training, tuning, and evaluation.

2.3 Initial Project Planning

Tools: Python, TensorFlow, Keras, Google Colab

Dataset: Custom dataset of 30 classes extracted from ZIP in Google Drive

Output: Trained .keras model and performance plots saved in Google Drive

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan and Raw Data Sources Identified

- **Source:** Manually compiled dataset stored in Google Drive (merged.zip)
- **Classes:** 30 fruit and vegetable types
- **Structure:** Split into train and validate folders

 YUDHA ISLAMI SULISTYA · UPDATED 2 YEARS AGO

91

<> Code

Download



Plants Type Datasets

Explore a comprehensive collection of plant species data for research and analysis



 test	6/20/2025 11:40 AM	File folder
 train	6/20/2025 11:41 AM	File folder
 validate	6/20/2025 11:41 AM	File folder

3.2 Data Quality Report

- Image counts were imbalanced across some classes.
- Class weights were computed using `sklearn.utils.class_weight` to address imbalance.

3.3 Data Preprocessing

- Resizing all images to **300×300** pixels
- Applied normalization, rotation, zoom, flipping, and shift transformations
- Created train and validation data generators with `ImageDataGenerator`

```

train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    zoom_range=0.2,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_gen = train_datagen.flow_from_directory(
    train_dir,
    target_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=True
)

val_gen = val_datagen.flow_from_directory(
    val_dir,
    target_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)

# Save class label mapping
with open(CLASS_INDICES_PATH, "w") as f:
    json.dump(train_gen.class_indices, f)

# Compute class weights
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.arange(len(train_gen.class_indices)),
    y=train_gen.classes
)

class_weights = dict(enumerate(class_weights))

```


4. Model Development Phase

4.1 Model Selection Report

- **Model:** EfficientNetB3 (pretrained on ImageNet)
- Chosen for its balance between accuracy and computational efficiency
- Pooling: Global average pooling
- Added Dense layer (256 units, ReLU), Dropout (0.4), and final softmax layer

```
def build_model():
    base_model = EfficientNetB3(
        include_top=False,
        weights='imagenet',
        input_shape=(*TARGET_SIZE, 3),
        pooling='avg'
    )
    base_model.trainable = True # Unfreeze from beginning

    x = layers.Dense(256, activation='relu')(base_model.output)
    x = layers.Dropout(0.4)(x)
    output = layers.Dense(train_gen.num_classes, activation='softmax')(x)

    model = models.Model(inputs=base_model.input, outputs=output)
    return model

model = build_model()
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy', tf.keras.metrics.TopKCategoricalAccuracy(k=3)]
)
```

4.2 Initial Model Training Code, Model Validation and Evaluation Report

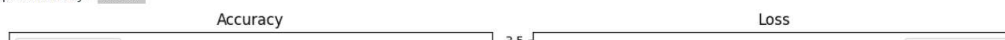
- **Optimizer:** Adam with learning rate 1e-5
- **Loss:** Categorical Crossentropy
- **Metrics:** Accuracy, Top-3 Accuracy
- **Epochs:** 15
- **Callbacks:** EarlyStopping and ModelCheckpoint used
- **Results:**
 - Validation Accuracy: 90.67%
 - Top-3 Accuracy: 98.22%

```

142/142 - 134s - 943ms/step - accuracy: 0.2210 - loss: 2.9895 - top_k_categorical_accuracy: 0.4140 - val_accuracy: 0.2089 - val_loss: 3.0925 - val_top_k_categorical_accuracy: 0.3778
Epoch 4/15
142/142 - 131s - 920ms/step - accuracy: 0.3419 - loss: 2.6856 - top_k_categorical_accuracy: 0.5589 - val_accuracy: 0.4489 - val_loss: 2.5738 - val_top_k_categorical_accuracy: 0.6778
Epoch 5/15
142/142 - 130s - 917ms/step - accuracy: 0.4508 - loss: 2.3475 - top_k_categorical_accuracy: 0.6802 - val_accuracy: 0.4667 - val_loss: 2.3783 - val_top_k_categorical_accuracy: 0.7111
Epoch 6/15
142/142 - 137s - 966ms/step - accuracy: 0.5357 - loss: 2.0081 - top_k_categorical_accuracy: 0.7644 - val_accuracy: 0.6689 - val_loss: 1.7883 - val_top_k_categorical_accuracy: 0.8667
Epoch 7/15
142/142 - 144s - 1s/step - accuracy: 0.6138 - loss: 1.7005 - top_k_categorical_accuracy: 0.8277 - val_accuracy: 0.7089 - val_loss: 1.4883 - val_top_k_categorical_accuracy: 0.8956
Epoch 8/15
142/142 - 128s - 903ms/step - accuracy: 0.6751 - loss: 1.4385 - top_k_categorical_accuracy: 0.8591 - val_accuracy: 0.7289 - val_loss: 1.3410 - val_top_k_categorical_accuracy: 0.9089
Epoch 9/15
142/142 - 140s - 982ms/step - accuracy: 0.7111 - loss: 1.2323 - top_k_categorical_accuracy: 0.8950 - val_accuracy: 0.7267 - val_loss: 1.1942 - val_top_k_categorical_accuracy: 0.9200
Epoch 10/15
142/142 - 126s - 885ms/step - accuracy: 0.7371 - loss: 1.0877 - top_k_categorical_accuracy: 0.9045 - val_accuracy: 0.8333 - val_loss: 0.7962 - val_top_k_categorical_accuracy: 0.9689
Epoch 11/15
142/142 - 130s - 916ms/step - accuracy: 0.7764 - loss: 0.9317 - top_k_categorical_accuracy: 0.9257 - val_accuracy: 0.8378 - val_loss: 0.7208 - val_top_k_categorical_accuracy: 0.9711
Epoch 12/15
142/142 - 131s - 924ms/step - accuracy: 0.7858 - loss: 0.8550 - top_k_categorical_accuracy: 0.9263 - val_accuracy: 0.8467 - val_loss: 0.6462 - val_top_k_categorical_accuracy: 0.9711
Epoch 13/15
142/142 - 129s - 911ms/step - accuracy: 0.8136 - loss: 0.7493 - top_k_categorical_accuracy: 0.9393 - val_accuracy: 0.8800 - val_loss: 0.5133 - val_top_k_categorical_accuracy: 0.9778
Epoch 14/15
142/142 - 138s - 969ms/step - accuracy: 0.8152 - loss: 0.6894 - top_k_categorical_accuracy: 0.9493 - val_accuracy: 0.8778 - val_loss: 0.4802 - val_top_k_categorical_accuracy: 0.9822
Epoch 15/15
142/142 - 128s - 899ms/step - accuracy: 0.8447 - loss: 0.6116 - top_k_categorical_accuracy: 0.9579 - val_accuracy: 0.9067 - val_loss: 0.4189 - val_top_k_categorical_accuracy: 0.9822
15/15 ----- 19s 465ms/step - accuracy: 0.8865 - loss: 0.4397 - top_k_categorical_accuracy: 0.9833

```

★ Validation Accuracy: 90.67%
 ★ Top-3 Accuracy: 98.22%



```

# =====
# 6. TRAINING
# =====
print("🔍 Using GPU:", tf.test.gpu_device_name())
history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=EPOCHS,
    callbacks=callbacks,
    verbose=2,
    class_weight=class_weights
)

```

```

# =====
# 7. EVALUATION
# =====
model = tf.keras.models.load_model(MODEL_PATH)
val_loss, val_acc, val_top3 = model.evaluate(val_gen)
print(f"\n★ Validation Accuracy: {val_acc:.2%}")
print(f"★ Top-3 Accuracy: {val_top3:.2%}")

```

5. Model Optimization and Tuning Phase

5.1 Tuning Documentation

- Used class weights to manage data imbalance
- Fine-tuned entire EfficientNetB3 from start
- Adjusted learning rate (low at 1e-5 to ensure stability)
- Used data augmentation strategies to prevent overfitting

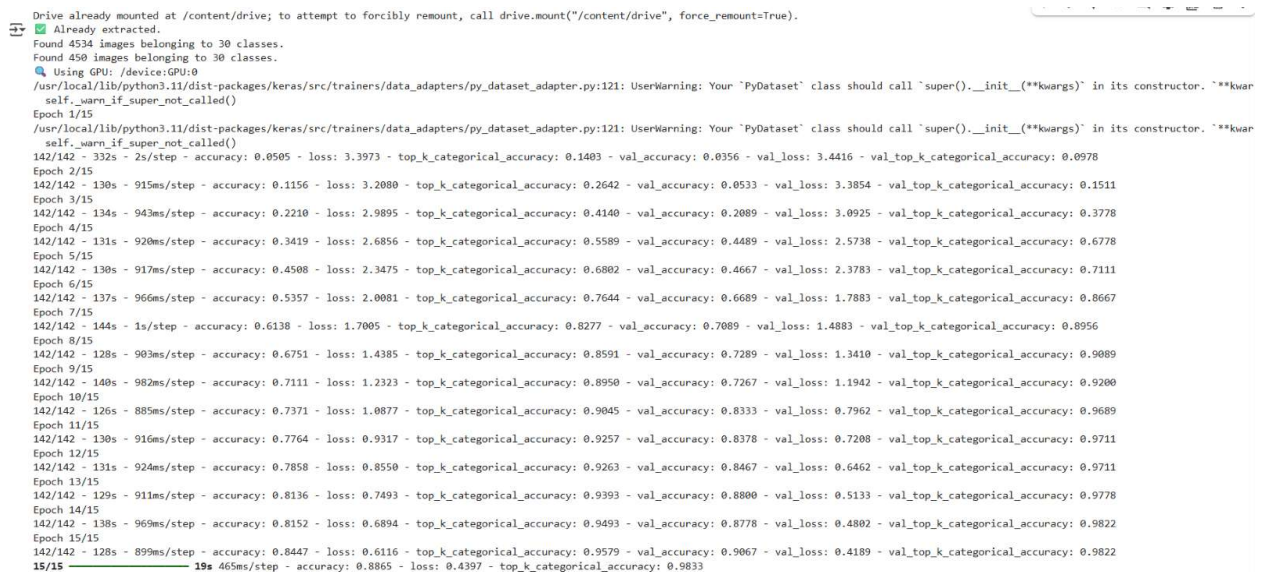
5.2 Results

- Improved validation accuracy with data augmentation and fine-tuning
- EarlyStopping helped avoid overfitting
- Model saved in .keras format to Drive

6. Final Model Selection Justification

EfficientNetB3 provided a strong balance between performance and efficiency. Fine-tuning the entire model led to better generalization. Lightweight for deployment and robust under real-world conditions.

7. Output Screenshots

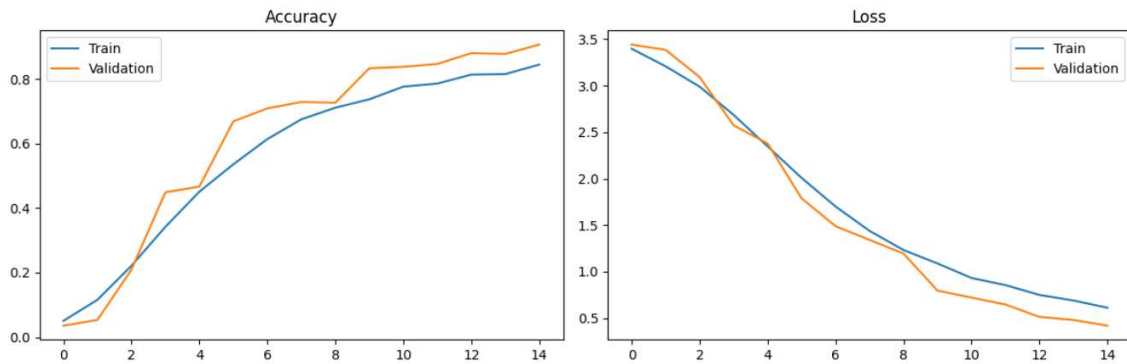


```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
[+] Already extracted.
Found 4534 images belonging to 30 classes.
Found 450 images belonging to 30 classes.
Using GPU: /device:GPU:0
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your 'PyDataset' class should call 'super().__init__(**kwargs)' in its constructor.
self._warn_if_super_not_called()
Epoch 1/15
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your 'PyDataset' class should call 'super().__init__(**kwargs)' in its constructor.
self._warn_if_super_not_called()
142/142 - 332s - 2s/step - accuracy: 0.0505 - loss: 3.3973 - top_k_categorical_accuracy: 0.1403 - val_accuracy: 0.0356 - val_loss: 3.4416 - val_top_k_categorical_accuracy: 0.0978
Epoch 2/15
142/142 - 130s - 915ms/step - accuracy: 0.1156 - loss: 3.2080 - top_k_categorical_accuracy: 0.2642 - val_accuracy: 0.0533 - val_loss: 3.3854 - val_top_k_categorical_accuracy: 0.1511
Epoch 3/15
142/142 - 134s - 943ms/step - accuracy: 0.2210 - loss: 2.9895 - top_k_categorical_accuracy: 0.4140 - val_accuracy: 0.2089 - val_loss: 3.0925 - val_top_k_categorical_accuracy: 0.3778
Epoch 4/15
142/142 - 131s - 920ms/step - accuracy: 0.3419 - loss: 2.6856 - top_k_categorical_accuracy: 0.5589 - val_accuracy: 0.4489 - val_loss: 2.5738 - val_top_k_categorical_accuracy: 0.6778
Epoch 5/15
142/142 - 130s - 917ms/step - accuracy: 0.4508 - loss: 2.3475 - top_k_categorical_accuracy: 0.6802 - val_accuracy: 0.4667 - val_loss: 2.3783 - val_top_k_categorical_accuracy: 0.7111
Epoch 6/15
142/142 - 137s - 966ms/step - accuracy: 0.5357 - loss: 2.0081 - top_k_categorical_accuracy: 0.7644 - val_accuracy: 0.6689 - val_loss: 1.7883 - val_top_k_categorical_accuracy: 0.8667
Epoch 7/15
142/142 - 144s - 1s/step - accuracy: 0.6138 - loss: 1.7005 - top_k_categorical_accuracy: 0.8277 - val_accuracy: 0.7089 - val_loss: 1.4883 - val_top_k_categorical_accuracy: 0.8956
Epoch 8/15
142/142 - 128s - 903ms/step - accuracy: 0.6751 - loss: 1.4385 - top_k_categorical_accuracy: 0.8591 - val_accuracy: 0.7289 - val_loss: 1.3410 - val_top_k_categorical_accuracy: 0.9089
Epoch 9/15
142/142 - 140s - 982ms/step - accuracy: 0.7111 - loss: 1.2323 - top_k_categorical_accuracy: 0.8950 - val_accuracy: 0.7267 - val_loss: 1.1942 - val_top_k_categorical_accuracy: 0.9200
Epoch 10/15
142/142 - 126s - 885ms/step - accuracy: 0.7371 - loss: 1.0877 - top_k_categorical_accuracy: 0.9045 - val_accuracy: 0.8333 - val_loss: 0.7962 - val_top_k_categorical_accuracy: 0.9689
Epoch 11/15
142/142 - 130s - 916ms/step - accuracy: 0.7764 - loss: 0.9317 - top_k_categorical_accuracy: 0.9257 - val_accuracy: 0.8378 - val_loss: 0.7208 - val_top_k_categorical_accuracy: 0.9711
Epoch 12/15
142/142 - 131s - 924ms/step - accuracy: 0.7858 - loss: 0.8550 - top_k_categorical_accuracy: 0.9263 - val_accuracy: 0.8467 - val_loss: 0.6462 - val_top_k_categorical_accuracy: 0.9711
Epoch 13/15
142/142 - 129s - 911ms/step - accuracy: 0.8136 - loss: 0.7493 - top_k_categorical_accuracy: 0.9393 - val_accuracy: 0.8800 - val_loss: 0.5133 - val_top_k_categorical_accuracy: 0.9778
Epoch 14/15
142/142 - 138s - 969ms/step - accuracy: 0.8152 - loss: 0.6894 - top_k_categorical_accuracy: 0.9493 - val_accuracy: 0.8778 - val_loss: 0.4802 - val_top_k_categorical_accuracy: 0.9822
Epoch 15/15
142/142 - 128s - 899ms/step - accuracy: 0.8447 - loss: 0.6116 - top_k_categorical_accuracy: 0.9579 - val_accuracy: 0.9067 - val_loss: 0.4189 - val_top_k_categorical_accuracy: 0.9822
15/15 - 465ms/step - accuracy: 0.8865 - loss: 0.4397 - top_k_categorical_accuracy: 0.9833

```

★ Validation Accuracy: 90.67%
★ Top-3 Accuracy: 98.22%



8. Advantages & Disadvantages

Advantages

- High accuracy and robust to lighting/occlusion issues
- Uses transfer learning (EfficientNet)
- Scalable for production use
- Easy integration with web/mobile via .keras model

Disadvantages

- Initial training is computationally expensive
- May misclassify highly similar-looking fruits (e.g., plums vs grapes)
- Requires good lighting for best results

9. Conclusion

The project successfully demonstrated how a deep learning model can automate the classification of fruits and vegetables. By leveraging transfer learning, we achieved strong performance with limited training data. The solution is scalable and relevant to real-world applications in agriculture and food supply chains.

10. Future Scope

- Build a mobile app for real-time fruit detection
- Integrate with a web dashboard for farm supervisors
- Expand dataset to include diseased fruits for health detection
- Add multilingual voice output for farmer support

11. Appendix

10.1 Source Code

<https://colab.research.google.com/drive/1ae1nGqvhRWlCgHNFcUCZyClp3Ro9C8WG#scrollTo=2THO86kqnLHM>

10.2 GitHub Link

<https://github.com/amith2144/fruits-and-vegetables-identification/tree/main/Deepfruitveg%20Automated%20Fruit%20And%20Veg%20Identification%20project>

Project Demo Link

<https://drive.google.com/file/d/1k1-I5TB7jpXIVNCcOq4PHK0T4UkDIYNt/view?usp=sharing>