Fruits_Project_AI

May 12, 2025

1 Fruits and Vegetables Recognition System

This project uses a Convolutional Neural Network (CNN) to classify images of fruits and vegetables. The system is designed to support fast, real-time predictions and can be applied in areas like food logging, grocery automation, and educational tools.

Project documentation attached in the last markdown cell.

1.1 Importing Libraries

[ ]:
```
import os
import zipfile
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import cv2

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D,␣
 ↪MaxPooling2D, Rescaling
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import RMSprop
import random
from tensorflow.keras.utils import load_img, img_to_array
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import load_img, img_to_array
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score

import warnings
warnings.filterwarnings('ignore')
```

1.2 Loading the Data

[ ]:
```
# Mouting Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
# zip path
zip_path = '/content/drive/MyDrive/Fruits and Vegetables.zip'

# Navigating the zip file
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
  extract_path = '/content/extracted_files'
  os.makedirs(extract_path, exist_ok=True)
  zip_ref.extractall(extract_path)

os.listdir(extract_path)
```

[ ]:
```
['test', 'train', 'validation']
```

[ ]:
```
# Set random seeds for reproducibility
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

1.3 Image Data Loading

Training Data

[ ]:
```
# Training folder path
training_path = '/content/extracted_files/train'

# Loading from directory
training_data = tf.keras.utils.image_dataset_from_directory(
    training_path,
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=42,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
```

```
    crop_to_aspect_ratio=False,
    pad_to_aspect_ratio=False,
    data_format=None,
    verbose=True,
)
```

```
Found 3115 files belonging to 36 classes.
```

[ ]:
```
# Extrcting labels and file path
training_class_names = training_data.class_names

image_paths = []
labels = []

for class_name in training_class_names:
    class_path = os.path.join(training_path, class_name)
    for image_name in os.listdir(class_path):
        image_path = os.path.join(class_path, image_name)
        image_paths.append(image_path)
        labels.append(class_name)
```

[ ]:
```
# Converting to DataFrame
train_df = pd.DataFrame({
    'image_path': image_paths,
    'label': labels
})

train_df.head()
```

[ ]:

	image_path	label
0	/content/extracted_files/train/apple/Image_3.jpg	apple
1	/content/extracted_files/train/apple/Image_23.jpg	apple
2	/content/extracted_files/train/apple/Image_60.jpg	apple
3	/content/extracted_files/train/apple/Image_47.jpg	apple
4	/content/extracted_files/train/apple/Image_19.jpg	apple

[ ]:
```
# Number of images
train_df.shape[0]
```

[ ]:
```
3115
```

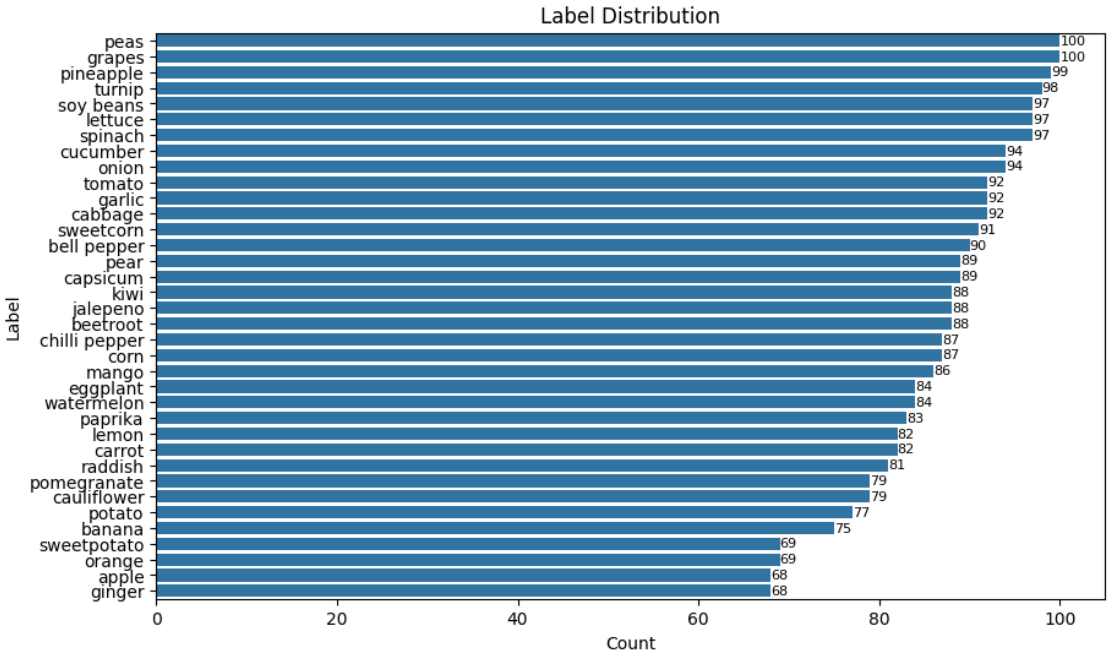
[ ]:
```
# duplicates
train_df.duplicated().sum()
```

[ ]:
```
np.int64(0)
```

[ ]:
```
# Missing values
train_df.isna().sum()
```

[ ]:
```
image_path    0
label         0
dtype: int64
```

[ ]:
```
# Bar Graph showing distribution of images in the 36 classed
label_counts = train_df['label'].value_counts()
plt.figure(figsize=(10, 6))
sns.barplot(y=label_counts.index, x=label_counts.values)

# data labels
for i, v in enumerate(label_counts.values):
    plt.text(v, i, str(v), color='black', ha='left', va='center', fontsize = 8)

# axes
plt.xlabel('Count')
plt.ylabel('Label')
plt.title('Label Distribution')
plt.show()
```



[ ]:
```
# Sample Images
plt.figure(figsize=(15,15))
```

```
for img_batch, label in training_data.take(1):
  for i in range(len(img_batch)):
    plt.subplot(8,8,i+1)
    plt.imshow(img_batch[i].numpy().astype('uint'))
    plt.axis('off')

    label_index = np.argmax(label[i].numpy())
    plt.title(training_class_names[label_index], fontsize = 7)

plt.show()
```

1.4 Validation Data

[ ]:
```
val_path = '/content/extracted_files/validation'

# Loading from directory
val_data = tf.keras.utils.image_dataset_from_directory(
    val_path,
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=42,
    validation_split=None,
    subset=None,
```

```
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False,
    pad_to_aspect_ratio=False,
    data_format=None,
    verbose=True,
)
```

```
Found 351 files belonging to 36 classes.
```

[ ]:
```
# Val class names
val_class_names = val_data.class_names

val_labels = []
val_image_paths = []

# Getting images
for val_class in val_class_names:
  val_class_path = os.path.join(val_path, val_class)
  for img in os.listdir(val_class_path):
    image_path = os.path.join(val_class_path, img)
    val_image_paths.append(image_path)
    val_labels.append(val_class)
```

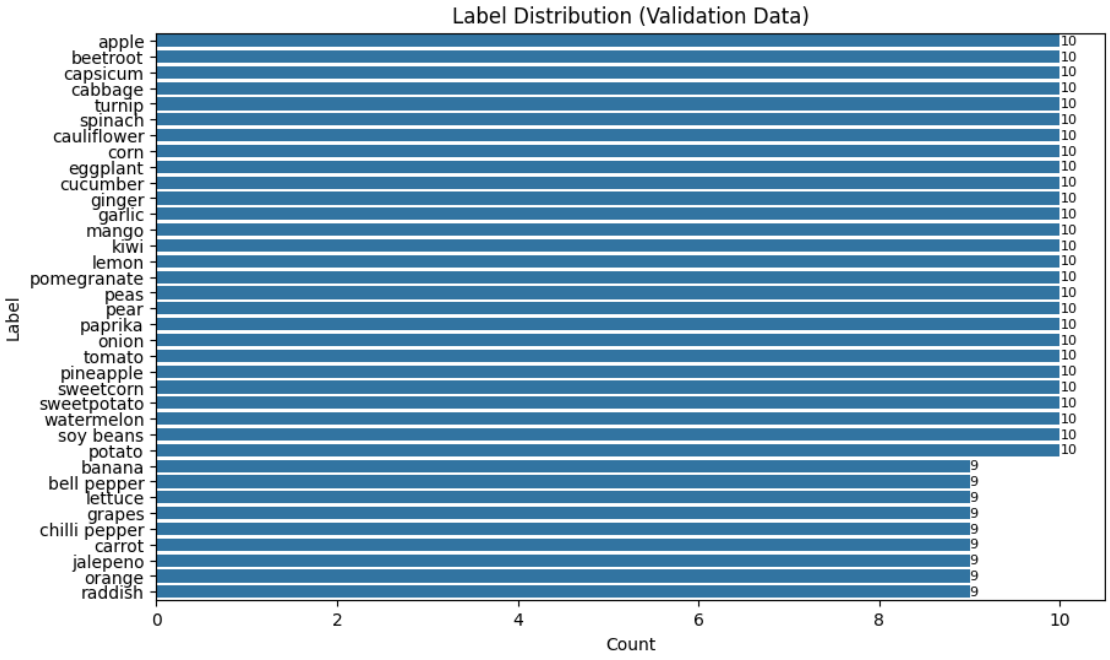
[ ]:
```
# data frame
val_df = pd.DataFrame({
    'image_path': val_image_paths,
    'label': val_labels
})

val_df.head()
```

[ ]:

	image_path	label
0	/content/extracted_files/validation/apple/Imag…	apple
1	/content/extracted_files/validation/apple/Imag…	apple
2	/content/extracted_files/validation/apple/Imag…	apple
3	/content/extracted_files/validation/apple/Imag…	apple
4	/content/extracted_files/validation/apple/Imag…	apple

[ ]:
```
val_lebls = val_df['label'].value_counts()

# Horizontal bar graph
plt.figure(figsize=(10, 6))
sns.barplot(y=val_lebls.index, x=val_lebls.values)

# data labels
for i, v in enumerate(val_lebls.values):
```

```
    plt.text(v, i, str(v), color='black', ha='left', va='center', fontsize = 8)

# axes
plt.ylabel('Label')
plt.xlabel('Count')
plt.title('Label Distribution (Validation Data)')
plt.show()
```



[ ]:
```
# Sample Data
plt.figure(figsize=(15,15))

for img_batch, label in val_data.take(1):
  for i in range(len(img_batch)):
    plt.subplot(8, 8, i+1)
    plt.imshow(img_batch[i].numpy().astype('uint'))
    plt.axis('off')

    label_index = np.argmax(label[i].numpy())
    plt.title(val_class_names[label_index])

plt.show()
```

1.5 Test Data

[ ]:
```
test_path = '/content/extracted_files/test'

# Loading from directory
test_data = tf.keras.utils.image_dataset_from_directory(
    test_path,
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size= (128, 128),
    shuffle=True,
    seed=42,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False,
    pad_to_aspect_ratio=False,
    data_format=None,
    verbose=True,
)
```

```
Found 359 files belonging to 36 classes.
```

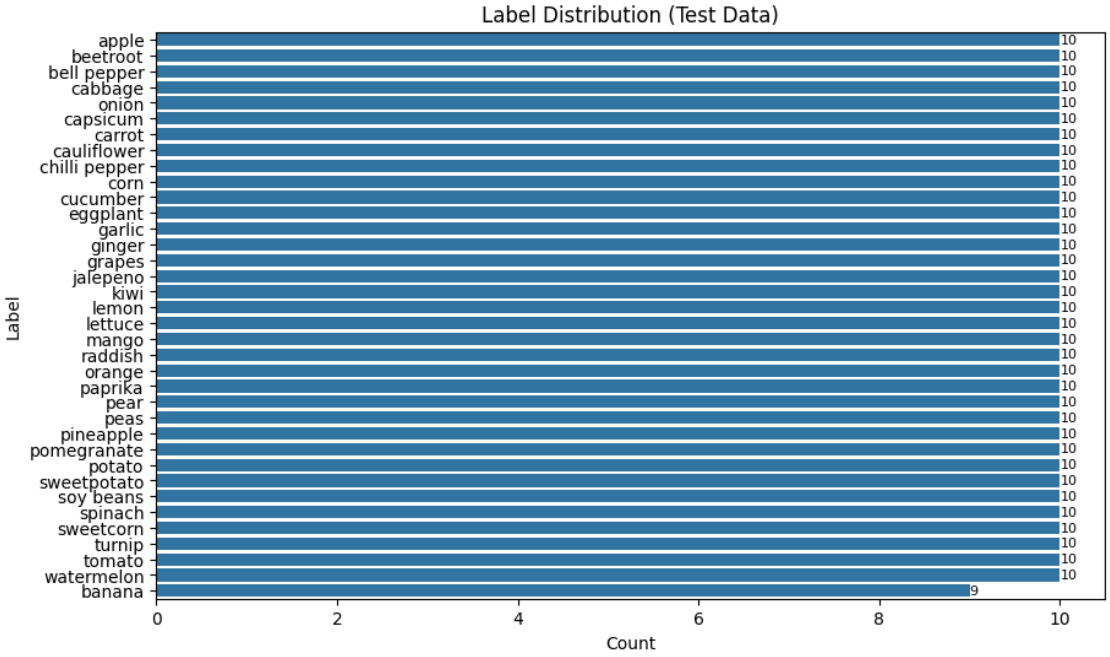
[ ]:
```
test_class_names = test_data.class_names

# Getting images
test_labels = []
test_image_paths = []

for test_class in test_class_names:
  test_class_path = os.path.join(test_path, test_class)
  for image in os.listdir(test_class_path):
    image_path = os.path.join(test_class_path, image)
    test_image_paths.append(image_path)
    test_labels.append(test_class)
```

[ ]:
```
# test_dataframe
test_df = pd.DataFrame({
    'image_path': test_image_paths,
    'label': test_labels
})

test_df.head()
```

[ ]:

	image_path	label
0	/content/extracted_files/test/apple/Image_3.jpg	apple
1	/content/extracted_files/test/apple/Image_10.jpg	apple
2	/content/extracted_files/test/apple/Image_1.jpg	apple
3	/content/extracted_files/test/apple/Image_2.jpg	apple
4	/content/extracted_files/test/apple/Image_6.JPG	apple

[ ]:
```
# distribution of labels in test dat
test_lebls = test_df['label'].value_counts()

# horizontal bar graph
plt.figure(figsize=(10, 6))
sns.barplot(y=test_lebls.index, x=test_lebls.values)

# data labels
for i, v in enumerate(test_lebls.values):
    plt.text(v, i, str(v), color='black', ha='left', va='center', fontsize = 8)

# axes
plt.ylabel('Label')
plt.xlabel('Count')
plt.title('Label Distribution (Test Data)')
plt.show()
```



[ ]:
```
# Sample Images in test Data
plt.figure(figsize=(15,15))

for img_batch, label in test_data.take(1):
  for i in range(len(img_batch)):
    plt.subplot(8, 8 , i+1)
    plt.imshow(img_batch[i].numpy().astype('uint'))
    plt.axis('off')

    label_index = np.argmax(label[i].numpy())
    plt.title(test_class_names[label_index])

plt.show()
```

[ ]:
```
# combined data
data = pd.concat([train_df, val_df, test_df], axis=0)

data.shape[0]
print(f"Total of {data.shape[0]} images used!")
```

```
Total of 3825 images used!
```

1.6 Training the Model

[ ]:
```
cnn_model = Sequential()

# First block
cnn_model.add(tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu',␣
 ↪padding='same'))
cnn_model.add(tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu'))
cnn_model.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2))

# Second block
cnn_model.add(tf.keras.layers.Conv2D(64, kernel_size=3, activation='relu',␣
 ↪padding='same'))
cnn_model.add(tf.keras.layers.Conv2D(64, kernel_size=3, activation='relu'))
cnn_model.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2))

# Second block
cnn_model.add(tf.keras.layers.Conv2D(128, kernel_size=3, activation='relu',␣
 ↪padding='same'))
cnn_model.add(tf.keras.layers.Conv2D(128, kernel_size=3, activation='relu'))
```

```
cnn_model.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2))
```

[ ]:
```
# Flatten
cnn_model.add(tf.keras.layers.Flatten())
```

[ ]:
```
# Adding nuerons
cnn_model.add(tf.keras.layers.Dense(units = 512, activation='relu'))
cnn_model.add(tf.keras.layers.Dense(units = 256, activation='relu'))

# dropping
cnn_model.add(tf.keras.layers.Dropout(0.5))
```

[ ]:
```
# Ouput layer
cnn_model.add(Dense(units = 36, activation='softmax'))
```

[ ]:
```
# Optimizer setting
optimizer = RMSprop(learning_rate=0.0001)

# Compiling the model
cnn_model.compile(optimizer = optimizer,  loss = 'categorical_crossentropy',␣
 ↪metrics = ['accuracy'])
```

[ ]:
```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=2,␣
 ↪restore_best_weights=True)
```

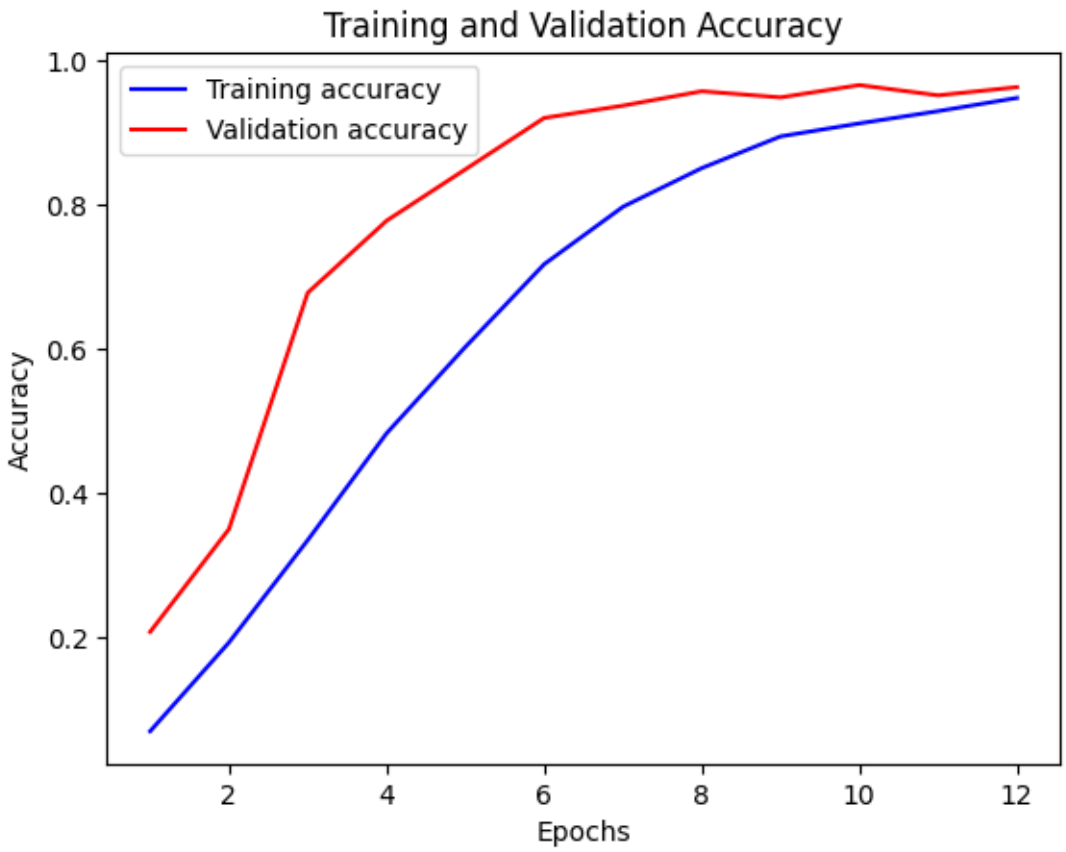
[ ]:
```
# Train the model
training_history = cnn_model.fit(x=training_data, validation_data = val_data,
                                 epochs = 25, verbose = 1,
                                 callbacks=[early_stopping] )
```

```
Epoch 1/25
98/98             67s 623ms/step -
accuracy: 0.0467 - loss: 4.8144 - val_accuracy: 0.2080 - val_loss: 2.8832
Epoch 2/25
98/98             76s 596ms/step -
accuracy: 0.1584 - loss: 3.0828 - val_accuracy: 0.3504 - val_loss: 2.2966
Epoch 3/25
98/98             82s 598ms/step -
accuracy: 0.3136 - loss: 2.4939 - val_accuracy: 0.6781 - val_loss: 1.2689
Epoch 4/25
98/98             60s 608ms/step -
accuracy: 0.4528 - loss: 1.9851 - val_accuracy: 0.7778 - val_loss: 0.9026
Epoch 5/25
98/98             82s 601ms/step -
accuracy: 0.5727 - loss: 1.5321 - val_accuracy: 0.8490 - val_loss: 0.6940
Epoch 6/25
```

```
98/98             58s 591ms/step -
accuracy: 0.6997 - loss: 1.0351 - val_accuracy: 0.9202 - val_loss: 0.4276
Epoch 7/25
98/98             58s 588ms/step -
accuracy: 0.7858 - loss: 0.7282 - val_accuracy: 0.9373 - val_loss: 0.2962
Epoch 8/25
98/98             83s 601ms/step -
accuracy: 0.8450 - loss: 0.5407 - val_accuracy: 0.9573 - val_loss: 0.2308
Epoch 9/25
98/98             82s 598ms/step -
accuracy: 0.8976 - loss: 0.3669 - val_accuracy: 0.9487 - val_loss: 0.2521
Epoch 10/25
98/98             77s 554ms/step -
accuracy: 0.9100 - loss: 0.3075 - val_accuracy: 0.9658 - val_loss: 0.1950
Epoch 11/25
98/98             85s 588ms/step -
accuracy: 0.9276 - loss: 0.2545 - val_accuracy: 0.9516 - val_loss: 0.2201
Epoch 12/25
98/98             57s 587ms/step -
accuracy: 0.9464 - loss: 0.2067 - val_accuracy: 0.9630 - val_loss: 0.2360
```

[ ]:
```
# training histroy df
training_history_df = pd.DataFrame(training_history.history)
```

[ ]:
```
# Plotting
epochs = range(1, len(training_history_df) + 1)
plt.plot(epochs, training_history_df['accuracy'], 'b', label='Training␣
 ↪accuracy')
plt.plot(epochs, training_history_df['val_accuracy'], 'r', label='Validation␣
 ↪accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



1.7 Saving the model

[ ]:
```
# saving the model
cnn_model.save('/content/drive/MyDrive/cnn_model.keras')
```

[ ]:
```
cnn_model.save('/content/cnn_model_fruits.keras')
```

1.8 Loading the model

[ ]:
```
cnn_model = load_model('/content/drive/MyDrive/fuits_model/cnn_model_fruits.
↪keras')
```

1.9 Model Evaluation

[ ]:
```
train_loss, train_accuracy = cnn_model.evaluate(training_data)
val_loss, val_accuracy = cnn_model.evaluate(val_data)
```

```
98/98              56s 534ms/step -
accuracy: 0.9904 - loss: 0.0666
```

```
11/11              11s 945ms/step -
accuracy: 0.9697 - loss: 0.1596
```

[ ]:
```
print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")

print("")
print(f"Training Loss: {train_loss:.4f}")
print(f"Validation Loss: {val_loss:.4f}")
```

```
Training Accuracy: 99.00%
Validation Accuracy: 96.58%

Training Loss: 0.0640
Validation Loss: 0.1950
```

1.10 Prediction on one image

[ ]:
```
image_test_path = '/content/extracted_files/test/carrot/Image_10.jpg'
img = cv2.imread(image_test_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.title('Test Image')
plt.axis('off')
plt.ylabel([])
plt.xlabel([])
plt.show()
```

[ ]:
```
image = load_img(image_test_path, target_size=(128, 128))

# converting image to array form
input_arr = img_to_array(image)

# converting input array into numpy
input_arr = np.array([input_arr])

# predicting
prediction = cnn_model.predict(input_arr)
```

```
1/1             0s 54ms/step
```

[ ]:
```
print(prediction[0])
print(max(prediction[0]))
```

```
[5.5998949e-05 3.2058815e-06 4.6437221e-06 7.4341833e-06 9.8635007e-07
 8.1716803e-07 9.9962878e-01 2.3672515e-05 5.3536610e-06 7.2934978e-07
 3.2791894e-07 3.3971656e-07 4.9462346e-06 2.0589637e-06 1.4747349e-07
 4.9479684e-07 1.8318084e-07 1.3123978e-06 1.2127421e-06 1.5871020e-06
 1.3044672e-05 4.9651721e-06 4.0071077e-06 2.1201026e-06 1.0114322e-06
 2.1196272e-06 1.1320305e-05 5.1871280e-05 4.1549134e-05 7.0426353e-07
 9.3029093e-06 2.1429403e-06 3.4660752e-05 2.6660769e-05 4.3689332e-05
 6.4186793e-06]
0.9996288
```

[ ]:
```
result_index = np.where(prediction[0] == max(prediction[0]))
result_index
```

[ ]:
```
(array([6]),)
```

[ ]:
```
print('Its {}'.format(test_class_names[result_index[0][0]]))
```

```
Its carrot
```

1.11 Test Data (Confusion Matrix)

[ ]:
```
test2_path = '/content/extracted_files/test'

# Loading from directory
test2_data = tf.keras.utils.image_dataset_from_directory(
    test_path,
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size= (128, 128),
```

```
    shuffle=False,
    seed=42,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False,
    pad_to_aspect_ratio=False,
    data_format=None,
    verbose=True,
)
```

```
Found 359 files belonging to 36 classes.
```

[ ]:
```
# class names
test2_class_names = test2_data.class_names
```

[ ]:
```
test_class_names
```

[ ]:
```
['apple',
 'banana',
 'beetroot',
 'bell pepper',
 'cabbage',
 'capsicum',
 'carrot',
 'cauliflower',
 'chilli pepper',
 'corn',
 'cucumber',
 'eggplant',
 'garlic',
 'ginger',
 'grapes',
 'jalepeno',
 'kiwi',
 'lemon',
 'lettuce',
 'mango',
 'onion',
 'orange',
 'paprika',
 'pear',
 'peas',
 'pineapple',
 'pomegranate',
 'potato',
```

```
 'raddish',
 'soy beans',
 'spinach',
 'sweetcorn',
 'sweetpotato',
 'tomato',
 'turnip',
 'watermelon']
```

[ ]:
```
# Predictions
y_pred = cnn_model.predict(test2_data)
y_pred.shape
```

```
12/12            7s 579ms/step
```

[ ]:
```
(359, 36)
```

[ ]:
```
predicted_categories = np.argmax(y_pred, axis=1)
predicted_categories
```

[ ]:
```
array([ 0,  0,  0,  0,  7,  0,  0,  0,  0, 15,  1,  1, 12,  1,  1,  1,  1,
        1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  5,  5,
        5,  5,  5,  3,  3,  5,  5,  5,  6,  6,  6,  6,  6,  6,  6,  6,  6,
        6,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  8,  8, 26,  8,  8,  8,
        8,  8,  8,  8,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9, 10, 10, 10,
       10, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
       12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 13, 13,
       13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 15, 15, 15, 15,
       15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 17,
       17, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, 18, 18,
       18, 18, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 20, 20, 20, 20, 20,
       20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 22, 22,
       22, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23,
       23, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 25, 25, 25, 25, 25, 25,
       25, 25, 25, 25, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 27, 27, 20,
       27, 27, 27, 27, 27, 11, 27, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28,
       29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 30, 30, 30, 30, 30, 30, 30,
       30, 30, 30, 31, 31, 31, 31, 31,  9, 31, 31, 31,  9, 32, 32, 32, 32,
       32, 21, 32,  8, 32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 34,
       34, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35, 35,
       35, 35])
```

[ ]:
```
for x,y in test2_data:
  print(y)
  break
```

```
tf.Tensor(
```

```
[[1. 0. 0. … 0. 0. 0.]
 [1. 0. 0. … 0. 0. 0.]
 [1. 0. 0. … 0. 0. 0.]
 …
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]], shape=(32, 36), dtype=float32)
```

[ ]:
```
true_categories = tf.concat([y for x,y in test2_data], axis=0)
true_categories
```

[ ]:
```
<tf.Tensor: shape=(359, 36), dtype=float32, numpy=
array([[1., 0., 0., …, 0., 0., 0.],
       [1., 0., 0., …, 0., 0., 0.],
       [1., 0., 0., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 0., 0., 1.],
       [0., 0., 0., …, 0., 0., 1.],
       [0., 0., 0., …, 0., 0., 1.]], dtype=float32)>
```

[ ]:
```
y_true = np.argmax(true_categories, axis=1)
y_true
```

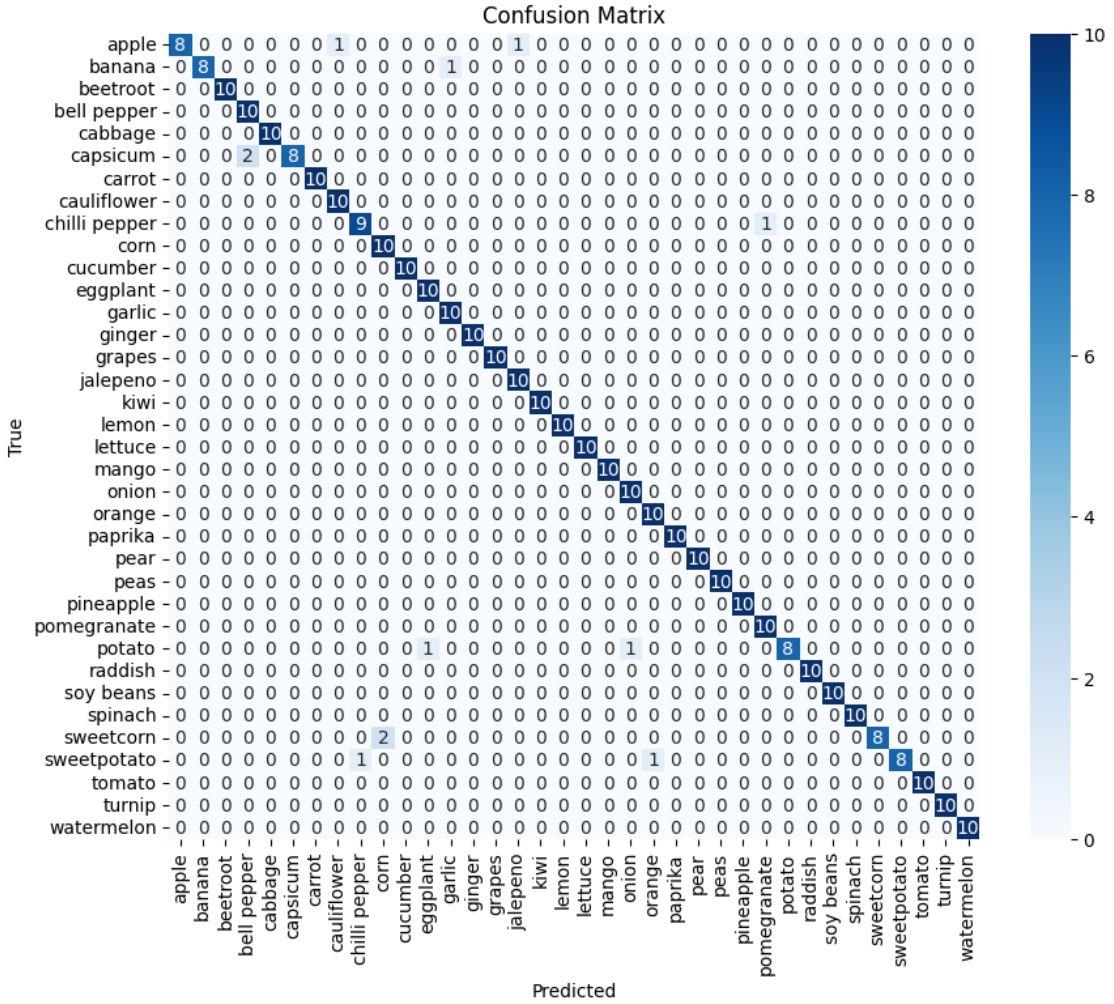
[ ]:
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  5,  5,
        5,  5,  5,  5,  5,  5,  5,  6,  6,  6,  6,  6,  6,  6,  6,  6,
        6,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  8,  8,  8,  8,  8,  8,
        8,  8,  8,  8,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9, 10, 10, 10,
       10, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
       12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 13, 13,
       13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 15, 15, 15, 15,
       15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 17,
       17, 17, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, 18,
       18, 18, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 20, 20, 20, 20, 20,
       20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 22, 22,
       22, 22, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 23, 23, 23, 23, 23,
       23, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 25, 25, 25, 25, 25, 25,
       25, 25, 25, 25, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 27, 27, 27,
       27, 27, 27, 27, 27, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28,
       29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 30, 30, 30, 30, 30, 30, 30,
       30, 30, 30, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32,
       32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 34,
       34, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35, 35,
       35, 35])
```

[ ]:
```
# Classification Report
cr = classification_report(y_true, predicted_categories,␣
 ↪target_names=test2_class_names)
print(cr)
```

```
                precision    recall  f1-score   support

         apple       1.00      0.80      0.89        10
        banana       1.00      0.89      0.94         9
      beetroot       1.00      1.00      1.00        10
    bell pepper       0.83      1.00      0.91        10
       cabbage       1.00      1.00      1.00        10
      capsicum       1.00      0.80      0.89        10
        carrot       1.00      1.00      1.00        10
    cauliflower       0.91      1.00      0.95        10
  chilli pepper       0.90      0.90      0.90        10
          corn       0.83      1.00      0.91        10
      cucumber       1.00      1.00      1.00        10
      eggplant       0.91      1.00      0.95        10
        garlic       0.91      1.00      0.95        10
        ginger       1.00      1.00      1.00        10
        grapes       1.00      1.00      1.00        10
      jalepeno       0.91      1.00      0.95        10
          kiwi       1.00      1.00      1.00        10
         lemon       1.00      1.00      1.00        10
       lettuce       1.00      1.00      1.00        10
         mango       1.00      1.00      1.00        10
         onion       0.91      1.00      0.95        10
        orange       0.91      1.00      0.95        10
       paprika       1.00      1.00      1.00        10
          pear       1.00      1.00      1.00        10
          peas       1.00      1.00      1.00        10
     pineapple       1.00      1.00      1.00        10
    pomegranate       0.91      1.00      0.95        10
        potato       1.00      0.80      0.89        10
       raddish       1.00      1.00      1.00        10
     soy beans       1.00      1.00      1.00        10
       spinach       1.00      1.00      1.00        10
     sweetcorn       1.00      0.80      0.89        10
    sweetpotato       1.00      0.80      0.89        10
        tomato       1.00      1.00      1.00        10
        turnip       1.00      1.00      1.00        10
    watermelon       1.00      1.00      1.00        10

      accuracy                           0.97       359
     macro avg       0.97      0.97      0.97       359
  weighted avg       0.97      0.97      0.97       359
```

[ ]:
```
# Confusion Matrix
cm = confusion_matrix(y_true, predicted_categories)
print(cm.shape)

# Visualizing
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=test2_class_names, yticklabels=test2_class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
(36, 36)
```



1.12 Model Deployment

[ ]:
```
import streamlit as st
import numpy as np
import tensorflow as tf
import os
import gdown
from PIL import Image

# Loading the trained model
def get_model():
    model_path = "cnn_model_fruit.keras"
    file_id = "1AQi309SipD25u86nGWRqTTsYoJZBUEFj"
    url = f"https://drive.google.com/uc?id={file_id}"
    if not os.path.exists(model_path):
        gdown.download(url, model_path, quiet=False)
    return tf.keras.models.load_model(model_path)


# Model Prediction
def model_prediction(test_image):
    # load the model
    cnn_model = get_model()
    image = tf.keras.preprocessing.image.load_img(test_image, target_size=(128,␣
 ↪128))
    # converting image to array form
    input_arr = tf.keras.preprocessing.image.img_to_array(image)
    # converting input array into numpy / batch
    input_arr = np.array([input_arr])
    # predicting
    predictions = cnn_model.predict(input_arr)
    # return index of max element
    return np.argmax(predictions)

# The App
st.sidebar.title('Dashboard')
app = st.sidebar.selectbox("Select Page", ["Home", "About Project",␣
 ↪"Predictions"])

# Home Page
if (app == "Home"):
    st.header("FOOD PREDICTION")
    st.write("""
        Welcome to the Food Prediction System!

        This tool uses a deep learning model to predict the type of fruit or␣
 ↪vegetable in an image you upload.
```

```
        It's designed to be fast, accurate, and easy to use - just exploring␣
 ↪computer vision technology.

        To get started, navigate to the **Predict** tab and upload an image of␣
 ↪a fruit or vegetable.
    """)

# About Page
if (app == "About Project"):
    st.header("About Project")
    st.write("""
        This Fruit and Vegetable Classification System uses a Convolutional␣
 ↪Neural Network (CNN) model to identify various fruits and vegetables from␣
 ↪images.
        The goal is to support quick and accurate food recognition, which can␣
 ↪be useful in applications such as:

        - Automated checkout systems in grocery stores
        - Dietary tracking and food logging apps
        - Educational tools for kids to learn about healthy foods
        - Agricultural produce classification

        The model has been trained on a diverse dataset of common fruits and␣
 ↪vegetables to ensure high prediction accuracy.
        Upload an image, and the system will classify it in real-time with its␣
 ↪corresponding label.
    """)

# Predictions Page
if (app == "Predictions"):
    st.header("Predictions")
    test_image = st.file_uploader("Chose an image")
    if (st.button("Show image")):
        img = Image.open(test_image)
        img = img.resize((250,250))
        st.image(img)

    # Predictions
    if (st.button("Predict")):
        st.write("The Prediction")
        result_index = model_prediction(test_image)
        # Labels
        with open(r"labels.txt") as f:
            content = f.readlines()
            label = []
            for i in content:
```

```
                label.append(i[:-1])
            st.success('Its {}'.format(label[result_index]))
```

1.13 Project Documentation

1.13.1 Data Preparation and Exploration

1. Mount Google Drive to access dataset.
2. Extract ZIP file containing train, validation, and test folders.

Data set exploration:

3. Counting the number of images belonging to each class present in the dataset.
4. Visualizing the distribution of labels across the dataset using bar plots.
5. Displaying sample images from each of the three dataset splits (train, validation, test) to understand the visual characteristics of the data.

1.13.2 Datasets Loading

1. Load training, validation, and test datasets using image_dataset_from_directory.
2. Normalize image size to (128x128) and set batch size to 32.
3. Label Inference: The class labels were automatically inferred based on the directory structure of the loaded datasets.
4. EDA DataFrames: Pandas DataFrames were created to store the file paths of the images and their corresponding labels. These DataFrames were used for further exploratory data analysis.

1.13.3 CNN Model Architecture:

A Sequential Convolutional Neural Network (CNN) model was constructed with the following architecture:

1. 3 Convolutional blocks (two Conv2D + one MaxPooling)
2. Fully connected Dense layers
3. Dropout to prevent overfitting
4. Output Layer: The final layer is a Softmax output layer, which is standard for multiclass classification tasks. It produces a probability distribution over the different classes.

1.13.4 Model Compilation and Training

Compiled the model using:

1. Optimizer: RMSprop optimizer, to update the model's weights during training.
2. Loss Function: Categorical cross-entropy loss, which is appropriate for multiclass classification problems where the labels are one-hot encoded.

3. EarlyStopping: Applied EarlyStopping to monitor validation loss and stop training when no improvement is seen. Model training automatically stopped early before 25 epochs based on validation performance.
4. Trained using both training and validation datasets.

1.13.5 Model Evaluation

1. Ploted training and validation accuracy/loss curves.
2. Evaluate the model on: Training set and Validation set.
3. Saved the trained model to Google Drive.

1.13.6 Single Image Prediction (Testing the Model)

To test the model's ability to predict the class of a single, unseen image:

1. A single image was loaded from the test set and resized to match the input dimensions expected by the model (128x128 pixels).
2. Converted the image into an array format and then to a NumPy array to form a batch.
3. The image's color format was confirmed to be RGB, consistent with the training data.
4. Passed the processed image to the trained CNN model to generate a prediction.
5. Displayed the image along with its predicted class label.

1.13.7 Bulk Testing and Evaluation

To obtain a comprehensive evaluation of the model's performance on the entire test set:

1. The test dataset was loaded again, but with shuffling disabled to ensure consistent order for evaluation.
2. Predictions were made for all images in the loaded test dataset.
3. A Classification Report was generated. This report provides key evaluation metrics for each class, includes precision, recall, and F1-score.
4. A Confusion Matrix was generated. This matrix provides a visual representation of the model's predictions, showing the counts of true positives, true negatives, false positives, and false negatives for each class. It offers insights into the types of errors the model makes.

1.13.8 Deployment

This Streamlit application offers a user-friendly web interface for fruit and vegetable classification.

It loads a pre-trained Keras CNN model from Google Drive, processes uploaded images to a 128x128 size, performs prediction, and displays the identified label using a labels.txt mapping.

The app features "Home", "About Project", and "Predictions" pages for easy navigation and interaction.

Link to App: Link